國立交通大學

電子工程學系　電子研究所碩士班

碩　士　論　文

適用於第三代無線行動通訊之雙模式通道解碼器的設計

A Dual Mode Channel Decoder for 3GPP2 Mobile Wireless Communications

學生 ： 施彥旭

指導教授 ： 李鎮宜 教授

中華民國九十三年六月

適用於第三代無線行動通訊之雙模式通道解碼器的設計

A Dual Mode Channel Decoder for 3GPP2 Mobile Wireless Communications

研究生：施彥旭　　Student：Yen-Hsu Shih

指導教授：李鎮宜　　Advisor：Chen-Yi Lee

國立交通大學
電子工程學系 電子研究所 碩士班
碩士論文

A Thesis
Submitted to Department of Electronics
College of Electrical Engineering and Computer Science
National Chiao Tung University
in Partial Fulfillment of the Requirements
for the Degree of
Master of Science
in

Electronics Engineering

June 2004

Hsinchu, Taiwan, Republic of China

中華民國九十三年六月

適用於第三代無線行動通訊之雙模式通道解碼器的設計

學生：施彥旭　　　　　　　　　　　指導教授：李鎮宜 教授

國立交通大學

電子工程學系 電子研究所碩士班

摘要

近年來，由於無線通訊應用的快速成長，前置錯誤更正碼的重要性與日俱增。其中，具有高解碼能力的迴旋碼與渦輪碼皆被廣泛運用。以第三代無線行動通訊標準為例，迴旋碼與渦輪碼同時被規範為其通道解碼器。以硬體實現的觀點而言，為兩者分別設計其專用的解碼器並不經濟。

本論文主旨即在設計一適用於第三代行動通訊標準之雙模式通道解碼器。此設計同時支援最大區塊長度為20,730之渦輪解碼器及不同編碼率的維特比解碼器。其最大資料輸出率分別可達每秒4.52Mb及5.26Mb。為減少外部記憶體的存取次數，我們採用一快取記憶體作為輸入緩衝器。經由有效率的打散器設計，記憶體需求與控制單元複雜度均可有效縮小。本架構以Verilog硬體描述語言撰寫，合成後邏輯閘數大約為11萬5千個。使用0.18um製程實作晶片，晶片面積約11.56mm^2，經過儀器量測在100MHz的時脈速度可正常運作，其晶片面積僅11.56mm^2。以固定六次迴圈的渦輪碼解碼模式下，平均只需要83mW的功率消耗即可達到標準所規範的最大資料輸出率每秒3.09Mb。

1896

A Dual Mode Channel Decoder for 3GPP2 Mobile Wireless Communications

Student : Yen-Hsu Shih Advisor : Dr. Chen-Yi Lee

Institute of Electronics Engineering

National Chiao Tung University

ABSTRACT

In the recent years, forward error correction schemes is rising and flourishing due to widespread increase of wireless communication applications. Among these standards, turbo codes and convolutional codes are usually adopted at the same time because of their high error correcting ability. However, to design the hardware functional block for each decoder is inefficient. In this thesis, a unified turbo and Viterbi decoder architecture for 3GPP2 standard is presented. The turbo decoding with a maximum block length of 20,730 and Viterbi decoding with various code rates are implemented to provide maximum data rate of 4.52Mb/s and 5.26Mb/s respectively at a clock rate of 100MHz. The memory access is reduced by the input caching scheme, and the system complexity is lowered by the efficient interleaver design. This chip is fabricated in a 0.18 μm six-metal standard CMOS process. The chip die size is 3.4 x 3.4 mm^2 with the core size of 2.7 x 2.7 mm^2. It contains 115k gates excluding the embedded memory. The measured power dissipation is 83mW while working at the clock rate of 66MHz to decode a 3.1Mb/s turbo encoded data stream with six iterations.

誌謝

隨著鳳凰花開，轉眼間又到了畢業的季節。在這二年的碩士生涯中，首先我要向指導教授李鎮宜博士表達最誠摯的謝意。由於老師指導有方，讓我能在短時間內找到正確的研究方向；在遇到挫折時也能從經驗中學習，培養正確的研究精神。另外，我也要感謝 Si2 實驗室中的每一位成員。在這裡的每個人研究領域或有不同，但都願意彼此幫助，讓我不僅了解團隊工作的重要性，更令人倍感溫馨；尤其我要感謝林建青學長，在我研究過程中不厭其煩地提供不少建議。最後，我要謝謝在背後默默支持著我的家人和朋友，讓我順利完成了這份學業。在大家的鼓勵下，讓我過得更多采多姿，我一定不會忘記這段令人充滿回憶的生活。

Contents

CHAPTER 1 INTRODUCTION 1

1.1 MOTIVATION 1

1.2 DESIGN SPECIFICATION 3

1.3 THESIS ORGANIZATION 3

CHAPTER 2 TURBO CODING AND DECODING 4

2.1 PRINCIPLE OF TURBO CODEC 4

2.1.1 Turbo Encoding 4

2.1.2 Turbo Interleaver 6

2.1.3 Turbo Decoding 7

2.1.4 Error floor effect 9

2.2 MAP DECODING ALGORITHM FOR TURBO DECODING 10

2.2.1 The MAP algorithm 10

2.2.2 The Max-Log-MAP algorithm 14

2.2.3 The Log-MAP algorithm 15

2.2.4 SNR sensitivity of Max-Log-MAP and Log-MAP algorithm 17

2.3 SLIDING WINDOWED APPROACH 17

CHAPTER 3 PRINCIPLE OF CONVOLUTIONAL CODEC 20

3.1 CONVOLUTIONAL CODE 20

3.2 Viterbi Decoding 21
3.3 Trace-back Method 23
3.4 Summary 25

CHAPTER 4 FIXED POINT ANALYSIS OF DUAL MODE TURBO/VITERBI DECODER 26

4.1 Fixed Point Analysis for Turbo Decoder 26
4.1.1 Input LLR Representation 26
4.1.2 Extrinsic Data Representation 29
4.1.3 Bit width of Internal Variables 30
4.1.4 Performance under Fixed Point Simulation 34
4.2 Fixed Point Analysis of Viterbi Decoder 35
4.3 Summary 39

CHAPTER 5 ARCHITECTURE OF PROPOSED DUAL MODE TURBO/VITERBI DECODER 40

5.1 Architecture of Integrated Turbo/Viterbi Decoder 40
5.2 Architecture of Turbo Decoder 42
5.2.1 Single MAP Decoder design 43
5.2.2 Cache design 44
5.2.3 Transition Metric Unit (TMU) 45
5.2.4 ACS Unit 46
5.2.5 LLR unit 47
5.2.6 Interleaver design 48
5.3 Architecture of Viterbi Decoder 50
5.3.1 Transition Metric Unit (TMU) 51
5.3.2 Survivor Memory Management 52

1896

5.4 SUMMARY 55

CHAPTER 6 CHIP IMPLEMENTATION 56

6.1 CHIP SPECIFICATION 56

6.2 COMPARISON WITH OTHER SIMILAR WORK 58

6.3 SUMMARY 59

CHAPTER 7 CONCLUSION AND FUTURE WORK 60

7.1 CONCLUSION 60

7.2 FUTURE WORK 60

BIBLIOGRAPHY 61

List of Figures

FIG. 2.1 TURBO ENCODER FOR 3GPP2 STANDARD 5

FIG. 2.2 TRELLIS TERMINATION 5

FIG. 2.3 TURBO INTERLEAVER FOR 3GPP2 STANDARD 6

FIG. 2.4 TURBO DECODING FLOWCHART 8

FIG. 2.5 PERFORMANCE COMPARISON UNDER DIFFERENT ITERATION NUMBERS IN 3GPP2 STANDARD 9

FIG. 2.6 TRELLIS DIAGRAM OF TURBO CODE IN 3GPP2 STANDARD 11

FIG. 2.7 THE WINDOWED MAP ALGORITHM 18

FIG. 2.8: PERFORMANCE COMPARISON AMONG DIFFERENT SUB-BLOCK LENGTHS IN 3GPP2 STANDARD 19

FIG. 3.1 RATE 1/2 CONVOLUTIONAL ENCODER WITH THE CONSTRAINT LENGTH OF 9 20

FIG. 3.2 A SYSTEM PLATFORM OF THE CONVOLUTIONAL CODEC 21

FIG. 3.3 TRELLIS DIAGRAM OF THE (2, 1, 2) CONVOLUTIONAL CODE AND ITS SURVIVOR MEMORY 23

FIG. 3.4 TRACE-BACK PROCEDURE OF THE CONVOLUTIONAL CODE 24

FIG. 3.5 THE SIMULATION RESULT OF VITERBI DECODER UNDER DIFFERENT TRACE-BACK LENGTHS 25

FIG. 4.1 FIXED POINT SIMULATION RESULT OF THE INPUT SYMBOLS WITH BPSK MODULATION AND RATE 1/2 TURBO DECODING 28

FIG. 4.2 FIXED POINT SIMULATION RESULT OF THE INPUT SYMBOLS WITH 16-QAM MODULATION

AND RATE 1/5 TURBO DECODING 28

FIG. 4.3 FIXED POINT SIMULATION RESULT OF THE EXTRINSIC INFORMATION 30

FIG. 4.4 AN EIGHT-STATE TRELLIS DIAGRAM ILLUSTRATING MESSAGE PASSING WITHIN 3 TRELLIS SECTIONS. 32

FIG. 4.5 FIXED POINT ANALYSIS WITH DIFFERENT BIT-WIDTH OF PATH METRICS IN TURBO DECODING 34

FIG. 4.6 DESIGN LOSS OF FIXED POINT TURBO DECODER 35

FIG. 4.7 THE PERFORMANCE OF VITERBI DECODER WITH DIFFERENT QUANTIZATION LEVELS 36

FIG. 4.8 THE CONVERGENCE OF ANY TWO SURVIVOR PATHS IN VITERBI ALGORITHM 37

FIG. 4.9 THE PERFORMANCE OF VITERBI DECODER UNDER DIFFERENT BIT-WIDTHS OF PATH METRIC 38

FIG. 4.10 THE PERFORMANCE ANALYSIS ON SYSTEM PERFORMANCE FOR EACH KIND OF CODE RATE 38

FIG. 5.1 THE PROPOSED ARCHITECTURE OF INTEGRATED TURBO/VITERBI DECODER 41

FIG. 5.2 THE ARCHITECTURE OF INTEGRATED TURBO/VITERBI DECODER IN TURBO MODE 42

FIG. 5.3 A SINGLE MAP DECODER ARCHITECTURE FOR TURBO DECODING................................ 43

FIG. 5.4 THE INPUT CACHE ARCHITECTURE 44

FIG. 5.5 THE DETAIL TIMING CHART OF THE PROPOSED INPUT CACHE 45

FIG. 5.6 THE TMU ARCHITECTURE FOR TURBO DECODER 46

FIG. 5.7 THE ACS ARCHITECTURE FOR DUAL MODE TURBO/VITERBI DECODER 47

FIG. 5.8 THE LLR UNIT ARCHITECTURE FOR TURBO DECODER 48

FIG. 5.9 THE ARCHITECTURE OF SHARED MEMORY DESIGN IN TURBO DECODER 49

FIG. 5.10 THE ADDRESS GENERATOR FOR THE INTERLEAVER OF 3GPP2 TURBO DECODER 50

FIG. 5.11 THE ARCHITECTURE OF INTEGRATED TURBO/VITERBI DECODER IN VITERBI MODE 51

FIG. 5.12 THE ARCHITECTURE OF TMU CELL 52

FIG. 5.13 THE 3-POINTER EVEN ALGORITHM FOR SURVIVOR MEMORY MANAGEMENT 54

Fig. 5.14 The architecture of survivor memory management..54

Fig. 6.1 The microphoto of the decoder chip 57

List of Tables

TABLE 1.1: DIFFERENCE OF FEC SPECIFICATION BETWEEN 3GPP AND 3GPP2 STANDARDS........2

TABLE 1.2 SPECIFICATION OF CHANNEL CODING IN 3GPP2 STANDARD 3

TABLE 2.1 TURBO INTERLEAVER PARAMETERS 7

TABLE 4.1 SUMMARY OF BIT-WIDTH DECISION FOR TURBO MODE 39

TABLE 4.2 SUMMARY OF BIT-WIDTH DECISION FOR VITERBI MODE..39

TABLE 4.3 A COMPARISON OF BIT-WIDTH DECISION WITH [19] FOR TURBO MODE 39

TABLE 6.1 SUMMARY OF THE DECODER CHIP 57

TABLE 6.2 POWER MEASUREMENT RESULT OF THE DECODER CHIP...58

TABLE 6.3 COMPARISON WITH OTHER SIMILAR WORK 59

Chapter 1
Introduction

1.1 Motivation

In the last decade, the digital technologies were introduced for wireless communications to replace analogy system due to increased traffic, speech privacy, new services (data transmission), and robustness of transmission (enhanced coding technique). These include the global system of mobile communications (GSM) which is a mobile radio standard with the most subscribers worldwide. It adopts time-division multiple access (TDMA) system and provides the maximum data rate of 9.6 Kb/s. Obviously, this data rate cannot satisfy the demand of multimedia document transfer in 21st century. Besides, the number of customer is increasing much faster than expected. However, the TDMA system is a dimension-limited system. The number of dimensions is determined by the number of time slot. No additional users are allowed once all time slots are assigned. Hence, third generation mobile radio systems are proposed to meet the market requirement with higher rate data service and higher capacity.

1896

Up to now, Third Generation Partnership Projects 3GPP [1] and 3GPP2 [2] defined detail standard providing maximum data rate of about 2 Mb/s and 3 Mb/s, respectively. On the other hand, code division multiple access (CDMA) proposal was brought up due to its higher capacity. Unlike TDMA system, the number of CDMA channels depends on the level of total interference that can be tolerated in the system. Forward error correction code plays an important role here since it can improve the tolerance for interference and thus increase the capacity of CDMA system. In both 3GPP and 3GPP2 standards, turbo code and high complexity convolutional code ((3, 1, 8) and (6, 1, 8) respectively) are instituted so that transmission quality can be guaranteed.

In 3GPP2 mobile wireless communication standard, larger block length is specified in turbo code while comparing with 3GPP standard (about 4 times larger). Besides, the code rate of turbo code and convolutional code are reduced to 1/5 and 1/6 respectively. These make the integration of Viterbi decoder and turbo decoder more difficult due to higher complexity. Moreover, 3GPP2 standard provides higher data rate of up to 3.09 Mb/s, which indicates an intensive demand of memory bandwidth. The above-mentioned will cause more challenge while designing the unified decoder if embedded memory size and power dissipation issues are taken into account. Hence, we would like to address a solution for channel decoder compatible with 3GPP2 standard to solve both area and power problems by proposing a novel architecture to implement a dual mode turbo/Viterbi decoder. Table 1.1 shows the main difference of 3GPP and 3GPP2 standards in turbo code and convolutional code.

Table 1.1: Difference of FEC specification between 3GPP and 3GPP2 standards.

		3GPP	3GPP2
Turbo Code	Code Rate	1/3	1/5
	Maximum block length	5114	20730
Convolutional Code	Code Rate	1/2, 1/3	1/2, 1/3, 1/4, or 1/6
	Maximum Data Rate	2 Mb/s	3.09 Mb/s

1.2 *Design Specification*

Our objective is to design a turbo and Viterbi decoder single chip for 3GPP2 standard. The detail specification of turbo code and convolutional code is listed in Table 1.2.

Table 1.2 Specification of channel coding in 3GPP2 standard

Turbo Code	Constraint length	4
	Generator function	$\left[1 \quad \frac{1+D+D^3}{1+D^2+D^3} \quad \frac{1+D+D^2+D^3}{1+D^2+D^3}\right]$
	Required data rate	3.09 Mb/s
Convolutional Code	Constraint length	9
	Generator function (CR=1/6)	$[457, 755, 511, 637, 625, 727]_{(octal)}$
	Generator function (CR=1/4)	$[765, 671, 513, 473]_{(octal)}$
	Generator function (CR=1/3)	$[557, 663, 711]_{(octal)}$
	Generator function (CR=1/2)	$[753, 561]_{(octal)}$
	Required data rate	1.036 Mb/s

1.3 *Thesis Organization*

This thesis consists of 7 chapters. In chapter 2, we’ll focus on interpreting turbo coding and decoding algorithm and its relative techniques. The reader is assumed to be familiar with Viterbi algorithm and thus only a brief description is made in Chapter 3. Chapter 4 explains how we decide the fix-point resolution in our design. Some simulation results will also be shown here. In chapter 5, we present the proposed architecture. For clearness, operating flow for turbo mode and Viterbi mode will be discussed separately. In addition, several characteristic of our design will be stated here. Chapter 6 outlines the specification of our implemented chip. We also provide some comparisons with other similar works. Finally, conclusion and future work are made in chapter 7.

Chapter 2
Turbo Coding and Decoding

The parallel concatenated convolutional code (PCCC), named turbo code [3], was first proposed by C. Berrou, A. Glavieux, and P. Thitimajshima in 1993. It has been proved to have a performance close to Shannon limit with simple constituent codes concatenated by an interleaver. This new technique is now adopted in both 3GPP and 3GPP2 standards due to its excellent error correction ability. In this chapter, we'll describe the principle of both turbo encoding and turbo decoding methods. The error floor effect in turbo decoding and some decoding techniques will also be interpreted here.

2.1 Principle of Turbo codec

2.1.1 Turbo Encoding

The turbo encoder is composed of two recursive systematic convolutional (RSC) encoders, which are connected in parallel but separated by a turbo interleaver. The two RSC encoders are also called constituent codes of the turbo code. The block diagram of the turbo encoder is illustrated in Fig. 2.1. Note that the same input data are encoded by each RSC encoder but in different order. In 3GPP2 standard, each input bit is encoded as one systematic bit and two parity-check bits for each RSC encoder. Thus, the code rate of each component encoder is 1/3. In order to increase the code rate of turbo code, the systematic bits of the second RSC encoder are not transmitted. Therefore, the output encoded sequence should be $\{X, Y_0, Y_1, Y_0', Y_1'\}$, and the overall code rate is 1/5.

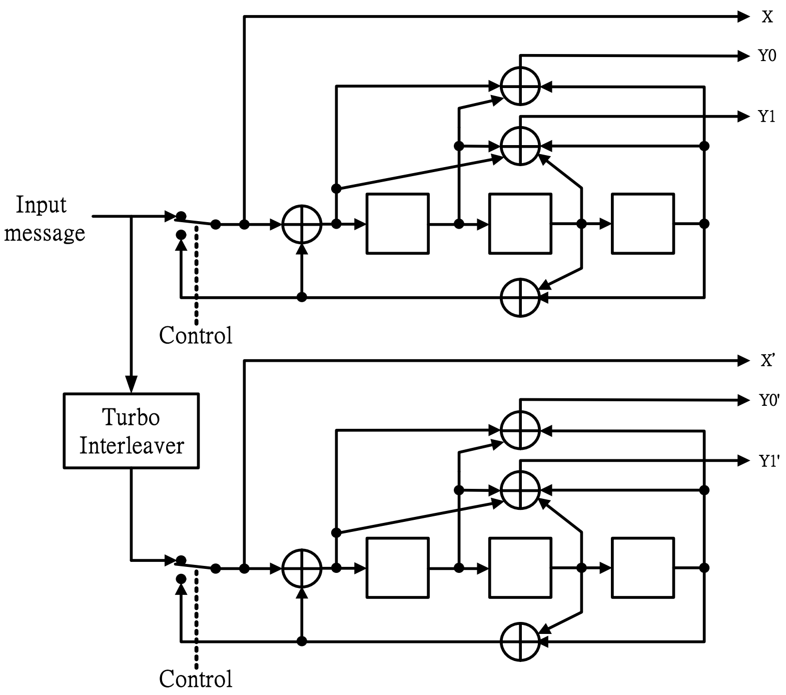


Fig. 2.1 Turbo encoder for 3GPP2 standard

After encoding all input messages, we have to generate several tail bits to set both component encoders back to zero state. However, it's impossible for a RSC encoder to return zero state by inserting dummy zeros into the encoder directly. Thus, a simple solution is provided in Fig. 2.2. While encoding input messages, the switch is set to position “A”. Once messages of whole block are encoded, the position of switch is changed to “B” for three additional cycles. This will force all registers to zeros and thus back to zero state.

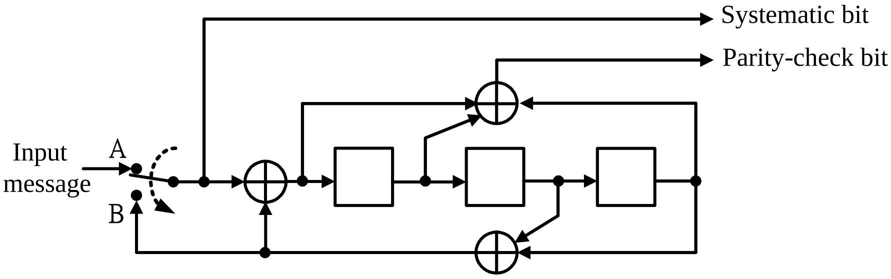


Fig. 2.2 Trellis Termination

2.1.2 Turbo Interleaver

The interleaver plays a very important role in turbo encoder. First of all, a proper coding gain can be achieved with small memory RSC encoders since the interleaver scrambles a long block message. Besides, the interleaver de-correlates the input of two RSC encoders so that iterative decoding algorithm can be applied between two component decoders. Theoretically, the block size of interleaver is one of the major factors to lower the upper bound on bit error probability of the turbo code system. The performance upper-bound of turbo code corresponding to a uniform random interleaver has been evaluated in [4]. The result shows that the bit-error-probability upper bound of turbo code is approximately proportional to 1/N, where N is the block size of turbo interleaver. The factor "1/N" is also called the interleaver gain.

Fig. 2.3 shows the address generator of turbo interleaver in 3GPP2 standard. It provides a maximum block size of 20,730 and minimum block size of 378. Detail supported block sizes and its corresponding "n" value are listed in Table 2.1.

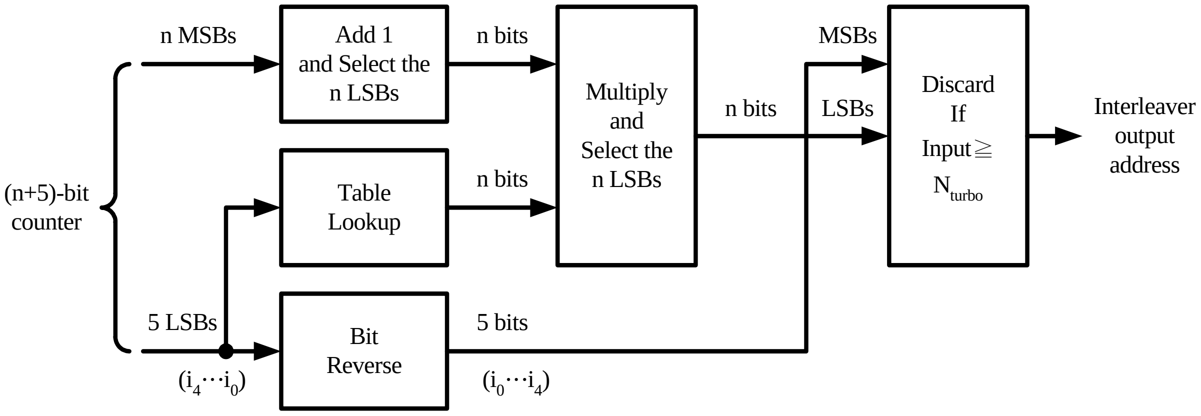


Fig. 2.3 Turbo Interleaver for 3GPP2 standard

Table 2.1 Turbo interleaver parameters

Turbo Interleaver Block size	Turbo interleaver parameter (n)
378	4
402	4
570	5
762	5
786	5
1,146	6
1,530	6
1,554	6
2,298	7
2,322	7
3,066	7
3,090	7
3,858	7
4,602	8
6,138	8
9,210	9
12,282	9
20,730	10

2.1.3 Turbo Decoding

A general idea for iterative turbo decoding is illustrated in Fig. 2.4, where r_s is the received systematic information, r_{p1} is the received parity information generated by the first

RSC encoder, and r_{p2} is the received parity information generated by the second RSC encoder. The iterative turbo decoding consists of two constituent decoders, which are soft-in/soft-out (SISO) decoders concatenated serially via one interleaver and one de-interleaver. An additional interleaver is used to interleave the input systematic information and then provides the interleaved data to the second SISO decoder. Two component decoders can be implemented based on either soft-output Viterbi algorithm (SOVA) [5] or maximum *a posteriori* probability (MAP) algorithm [6], which will be discussed particularly in the next section. During iterative decoding process, each constituent decoder delivers the extrinsic information $L_{ex}(u)$ which is taken as *a priori* information for the other constituent decoder. That is $L_{in1}(u_k) = \tilde{L}_{ex2}(u_k)$ and $L_{in2}(u_k) = \tilde{L}_{ex1}(u_k)$. As the number of iterations increases, better coding gain is expected. However, the correlation between two SISO decoders is also raised up. Therefore, there is no significant performance improvement if the number of iterations reaches a threshold. Fig. 2.5 shows the performance comparison under different iteration numbers in 3GPP2 standard.

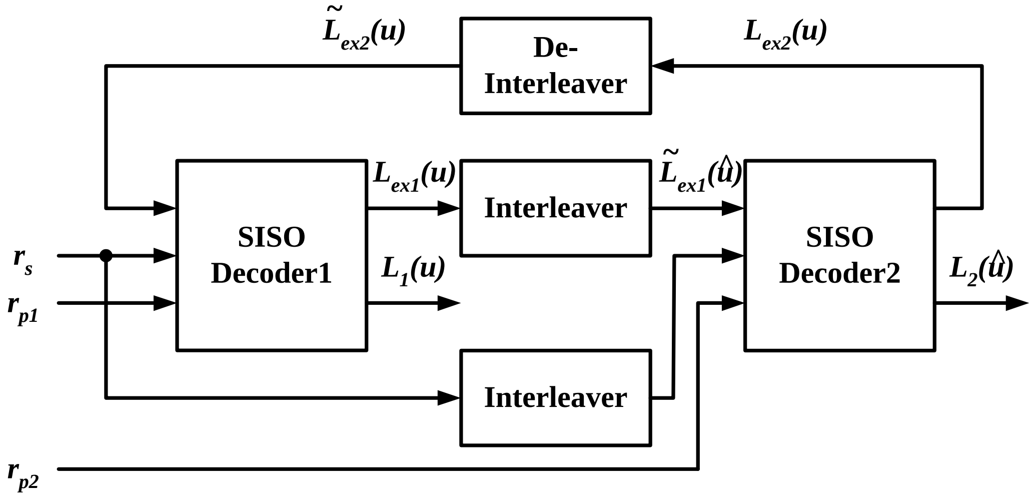


Fig. 2.4 Turbo decoding flowchart

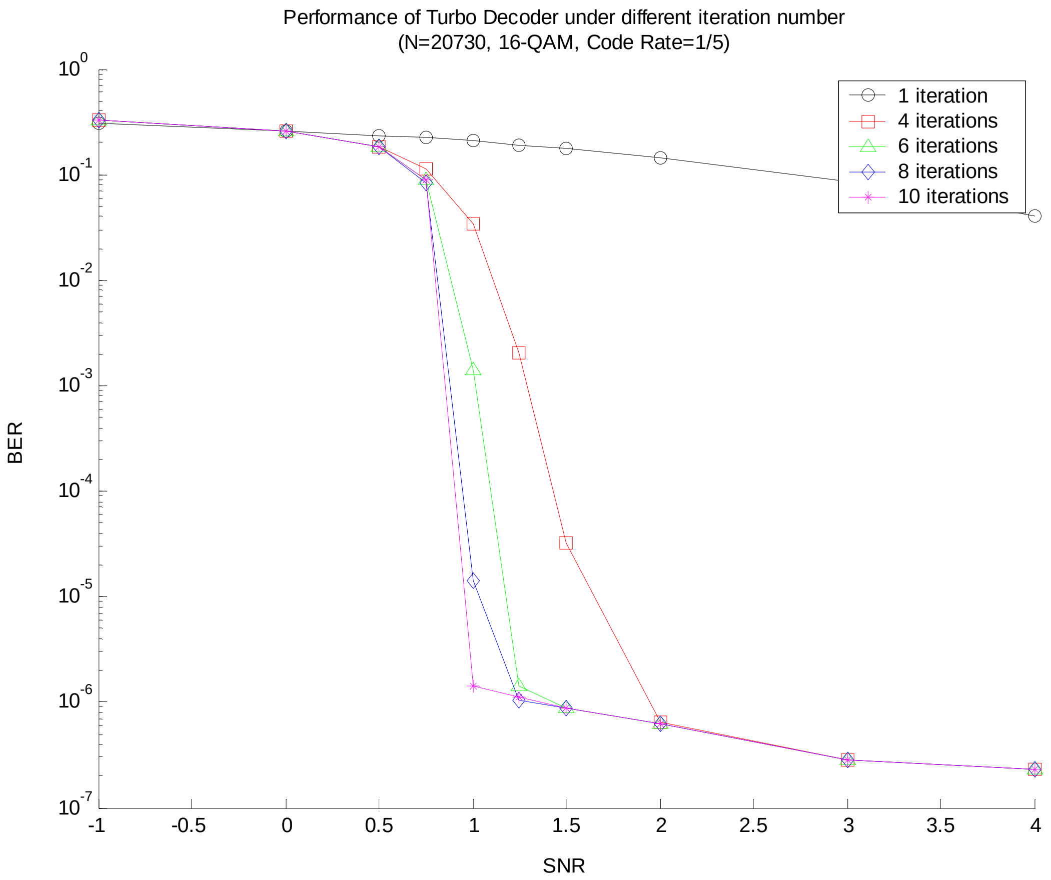


Fig. 2.5 Performance comparison under different iteration numbers in 3GPP2 standard

2.1.4 Error floor effect

Although turbo coding provides an excellent performance, the bit-error-rate certainly starts to decrease quite slowly at high signal-to-noise ratio (SNR). This phenomenon can be observed in Fig. 2.5. It is due to relative small free distance of turbo codes, and is called an “error floor” [7]. Consider the relation of the minimum free distance and the bit error probability in turbo coding, which can be expressed by

$$P_b \propto Q\left(\sqrt{2d_{free}R\frac{E_b}{N_0}}\right) \qquad (2.\ 1)$$

where d_{free} is the minimum free distance and E_b/N_0 is the SNR. At low SNR, the major part of errors can be corrected by iterative decoding since systematic information and parity information can be regarded as highly independent events. However, as the channel provides a reliable transmission, the dependency of the systematic and parity information grows up and the interleaver does little contribution on iterative decoding. Thus, the error correction ability is limited on the weak constituent code only. To overcome this issue, we can increase the interleaver size to lower the position of the error floor or concatenate a block code, e.g. BCH code, as an outer code to remove the left error bits. For more details, please refer to [4] [8].

2.2 MAP Decoding algorithm for Turbo Decoding

It has been proved that the MAP algorithm is the optimal decoding method for turbo code while comparing with SOVA [9]. Unlike Viterbi algorithm which utilizes maximum likelihood (ML) algorithm to find the codewords with minimum error probability, the MAP algorithm minimizes the symbol (or bit) error probability. In this section, we'll focus on introducing the turbo decoding methods based on MAP algorithm [6] [10]. Although SOVA is also one of the commonly used techniques for turbo decoding, we'll skip it since it's not adopted in our proposed design. To understand more detail about SOVA, please refer to [5]. And some comparisons of MAP algorithm and SOVA applied in turbo code system are shown in [9].

2.2.1 The MAP algorithm

The main idea of MAP algorithm is to compute the log-likelihood ratio (LLR) of the transmitted information bit u_k conditioned on the received information $\boldsymbol{r}_k$ for $1 \leqq k \leqq \mathrm{N}$, where N is the block length of encoded message.

$$L(\hat{u}_k) = L(u_k \mid \boldsymbol{r}) = \log \frac{P(u_k = +1 \mid \boldsymbol{r})}{P(u_k = -1 \mid \boldsymbol{r})} \tag{2. 2}$$

Here $\boldsymbol{r}$ is the vector of received soft values, and can be represented as $[r_1, r_2, \ldots, r_n]$ where n is the number of output bits for each encoded bit in the constituent code. Let's consider the trellis diagram of turbo code in 3GPP2 standard, which is shown in Fig. 2.6 as an example.

Note that the solid lines represent the transitions corresponding to an information bit u_k of -1, while the dotted lines represent the transitions corresponding to an information bit u_k of +1. Then, the equation can be further expressed as

$$L(\hat{u}_k) = \log \frac{P(u_k = +1 \mid \boldsymbol{r})}{P(u_k = -1 \mid \boldsymbol{r})} = \log \frac{\sum_{u_k=+1} P(s_{k-1}, s_k, \boldsymbol{r})}{\sum_{u_k=-1} P(s_{k-1}, s_k, \boldsymbol{r})} . \qquad (2.\ 3)$$

where the numerator and denominator are the sum of joint probabilities for all existing transitions from state s_{k-1} to state s_k that corresponding to an information bit u_k of +1 and -1 respectively.

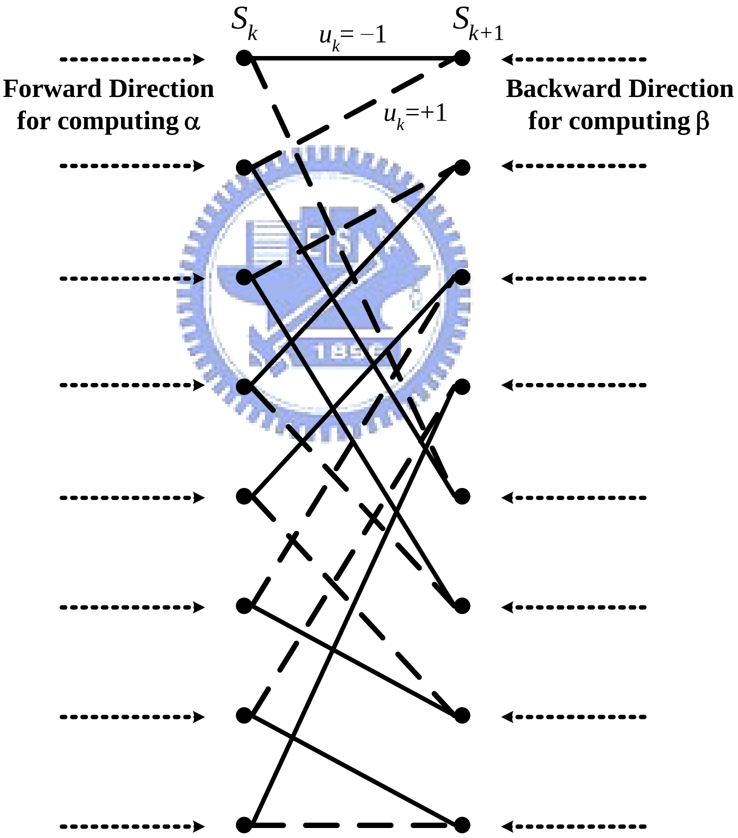


Fig. 2.6 Trellis diagram of turbo code in 3GPP2 standard

Assume the encoded data is transmitted through the discrete memoryless channel (DMC), and then the term $P(s_{k-1}, s_k, \boldsymbol{r})$ can be decomposed as three terms:

$$\begin{aligned} P(s_{k-1}, s_k, \boldsymbol{r}) &= \underbrace{P(s_{k-1}, r_{j<k})} \cdot \underbrace{P(s_k, \boldsymbol{r}_k \mid s_{k-1})} \cdot \underbrace{P(r_{j>k} \mid s_k)} \\ &= e^{\alpha_{k-1}(s_{k-1})} \cdot e^{\gamma_k(s_{k-1}, s_k)} \cdot e^{\beta_k(s_k)} \end{aligned} . \tag{2. 4}$$

Here $e^{\alpha_{k-1}(s_{k-1})}$ is the joint probability of state $s_{k\text{-}1}$ and received symbols $\boldsymbol{r}_j$ from the beginning of the block up to time index "*k*-1". Similarly, $e^{\beta_k(s_k)}$ is that of state s_k and received symbols $\boldsymbol{r}_j$ from the end of block back to time index "*k*". By shifting the value "*k*", it can be perceived that α is the forward recursion of the MAP algorithm, and can be formulated as

$$e^{\alpha_k(s_k)} = \sum_{s_{k-1}} e^{\gamma_k(s_{k-1}, s_k)} \cdot e^{\alpha_{k-1}(s_{k-1})} . \tag{2. 5}$$

The same as above, the backward recursion β can be formulated as

$$e^{\beta_{k-1}(s_{k-1})} = \sum_{s_k} e^{\gamma_k(s_{k-1}, s_k)} \cdot e^{\beta_k(s_k)} . \tag{2. 6}$$

Note that since the trellis of turbo code diverges from state zero and converges to state zero, the initial condition of the forward recursion and backward recursion should be set as

$$\begin{cases} e^{\alpha_0(s_0)} = 1, & \text{for } s_0 = 0 \\ e^{\alpha_0(s_0)} = 0, & \text{otherwise} \end{cases} \tag{2. 7}$$

and

$$\begin{cases} e^{\beta_N(s_N)} = 1, & \text{for } s_N = 0 \\ e^{\beta_N(s_N)} = 0, & \text{otherwise} \end{cases} \tag{2. 8}$$

For any existing transitions from $s_{k\text{-}1}$ to s_k, the branch transition probability $e^{\gamma_k(s_{k-1}, s_k)}$ can be further decomposed as

$$\begin{aligned} e^{\gamma_k(s_{k-1}, s_k)} &= P(s_k, \boldsymbol{r}_k \mid s_{k-1}) \\ &= P(s_k \mid s_{k-1}) \cdot P(\boldsymbol{r}_k \mid s_{k-1}, s_k) . \\ &= P(u_k) \cdot P(\boldsymbol{r}_k \mid u_k) \end{aligned} \tag{2. 9}$$

Here, the term "$P(u_k)$" is well-known as *a priori* probability. According to the definition of LLR, which is

$$L(u_k) = \log \frac{P(u_k = +1)}{P(u_k = -1)} , \tag{2. 10}$$

$P(u_k)$ can be rewritten as

$$\begin{aligned} P(u_k=\pm1) &= \frac{e^{\pm L(u_k)}}{1+e^{\pm L(u_k)}} \\ &= \frac{e^{-L(u_k)/2}}{1+e^{-L(u_k)}}\cdot e^{L(u_k)\cdot u_k/2} \\ &= A_k\cdot e^{L(u_k)\cdot u_k/2}. \end{aligned} \qquad (2.\ 11)$$

where the term A_k is equal for all transitions at the same time index, and thus will cancel out in (2. 3). On the other hand, the value of $P(\boldsymbol{r}_k|u_k)$ is dependent on channel characteristic. For an additive white Gaussian noise (AWGN) channel, the LLR of $\boldsymbol{r}_k$ conditioned on u_k can be expressed as

$$\begin{aligned} L(\boldsymbol{r}_k|u_k) &= \log\frac{P(\boldsymbol{r}_k\,|\,u_k=+1)}{P(\boldsymbol{r}_k\,|\,u_k=-1)} \\ &= \log\frac{\prod\limits_{\substack{v=1\\ u_k=+1}}^{n}\exp(-\frac{E_s}{N_0}(r_{k,v}-x_{k,v})^2)}{\prod\limits_{\substack{v=1\\ u_k=-1}}^{n}\exp(-\frac{E_s}{N_0}(r_{k,v}-x_{k,v})^2)} \\ &= \sum_{v=1}^{n}L_c\cdot r_{k,v}\cdot x_{k,v} \end{aligned} \qquad (2.\ 12)$$

where $L_c=4E_s/N_0$ and is called the channel reliability. Here, $x_{k,v}$ is the v-th transmitted symbol while encoding u_k. For systematic codes, $x_{k,1}$ is equal to u_k. Now we can obtain the value of $P(r_k|u_k)$ by using the technique in (2. 11) but substitute $L(u_k)$ with $L(r_k|u_k)$.

$$P(\boldsymbol{r}_k\,|\,u_k)=B_k\cdot\exp(\frac{1}{2}L_c r_{k,1}u_k+\frac{1}{2}\sum_{v=2}^{n}L_c r_{k,v}x_{k,v}) \qquad (2.\ 13)$$

For the same reason in (2. 11), B_k will also cancel out in (2. 3). Combining (2. 11) and (2. 13), the γ_k in (2. 9) can be reduced to

$$e^{\gamma_k(s_{k-1},s_k)}=A_k\cdot B_k\cdot\exp\left(\frac{1}{2}\left((L_c r_{k,1}+L(u_k))\cdot u_k+\sum_{v=2}^{n}L_c r_{k,v}x_{k,v}\right)\right). \qquad (2.\ 14)$$

Substituting (2. 5), (2. 6), (2. 14) into (2. 4), we can derive the *a posteriori* LLR in the form of

$$L(\hat{u}_k) = \log \frac{\sum\limits_{\substack{(s_{k-1},s_k)\\ u_k=+1}} e^{\alpha_{k-1}(s_{k-1})} \cdot e^{\gamma_k(s_{k-1},s_k)} \cdot e^{\beta_k(s_k)}}{\sum\limits_{\substack{(s_{k-1},s_k)\\ u_k=-1}} e^{\alpha_{k-1}(s_{k-1})} \cdot e^{\gamma_k(s_{k-1},s_k)} \cdot e^{\beta_k(s_k)}} \qquad (2.\ 15)$$

$$= L_c r_{k,1} + L(u_k) + L_{ex}(u_k)$$

where

$$L_{ex}(u_k) = \log \frac{\sum\limits_{\substack{(s_{k-1},s_k)\\ u_k=+1}} e^{\alpha_{k-1}(s_{k-1})} \cdot e^{\frac{1}{2}\sum_{v=2}^{n} L_c r_{k,v} x_{k,v}} \cdot e^{\beta_k(s_k)}}{\sum\limits_{\substack{(s_{k-1},s_k)\\ u_k=-1}} e^{\alpha_{k-1}(s_{k-1})} \cdot e^{\frac{1}{2}\sum_{v=2}^{n} L_c r_{k,v} x_{k,v}} \cdot e^{\beta_k(s_k)}} . \qquad (2.\ 16)$$

The term $L_{ex}(u_k)$ is called extrinsic information since it's a function of the redundant information that comes from the encoder. It removes the information about the systematic input and *a priori* information from $L(\hat{u}_k)$. Therefore, this term is useful to estimate *a priori* probability for the next component decoder, and great performance improvement in iterative MAP decoding can be achieved.

2.2.2 The Max-Log-MAP algorithm

As we can see, the MAP algorithm involves too many exponentiations and multiplications. These are quite complex for hardware realization. Thus, an approximation of MAP algorithm termed Max-Log-MAP algorithm [11] was derived for simple implementation of MAP decoders. Instead of calculating e^{γ_k}, e^{α_k}, and e^{β_k} directly, all computations are done in logarithm domain. Here we define γ_k, α_k, and β_k as transition metric, forward path metric and backward path metric respectively. γ_k can be formulated as

$$\gamma_k(s_{k-1}, s_k) = \log P(s_k, \boldsymbol{r}_k \mid s_{k-1}) . \qquad (2.\ 17)$$

Similarly, referring to (2. 4), α_k and β_k can be expressed as

$$\alpha_k(s_k) = \log P(s_k, \boldsymbol{r}_{j<k}) \qquad (2.\ 18)$$

and

$$\beta_{k-1}(s_{k-1}) = \log P(\mathbf{r}_{j>k} \mid s_k) \tag{2. 19}$$

respectively. After substituting (2. 17), (2. 18), and (2. 19), $L(\hat{u}_k)$ in (2. 15) can be re-written as

$$L(\hat{u}_k) = \log \frac{\sum\limits_{\substack{(s_{k-1},s_k)\\ u_k=+1}} \exp\left(\alpha_{k-1}(s_{k-1}) + \gamma_k(s_{k-1},s_k) + \beta_k(s_k)\right)}{\sum\limits_{\substack{(s_{k-1},s_k)\\ u_k=-1}} \exp\left(\alpha_{k-1}(s_{k-1}) + \gamma_k(s_{k-1},s_k) + \beta_k(s_k)\right)}. \tag{2. 20}$$

By utilizing the approximation of

$$\log(e^{\delta_1} + e^{\delta_2} + \cdots + e^{\delta_n}) \approx \max(\delta_1, \delta_2, \cdots, \delta_n), \tag{2. 21}$$

$L(\hat{u}_k)$ can be further simplified to

$$\begin{aligned} L(\hat{u}_k) = & \max_{\substack{(s_{k-1},s_k)\\ u_k=+1}} \left(\alpha_{k-1}(s_{k-1}) + \gamma_k(s_{k-1},s_k) + \beta_k(s_k)\right) \\ & - \max_{\substack{(s_{k-1},s_k)\\ u_k=-1}} \left(\alpha_{k-1}(s_{k-1}) + \gamma_k(s_{k-1},s_k) + \beta_k(s_k)\right). \end{aligned} \tag{2. 22}$$

This computation consists of forward and backward recursions that repetitively compute the α_k and β_k, and can be expressed by

$$\alpha_k(s_k) = \max_{s_{k-1},u_k}\left(\alpha_{k-1}(s_{k-1}) + \gamma_k(s_{k-1},s_k)\right) \tag{2. 23}$$

and

$$\beta_{k-1}(s_{k-1}) = \max_{s_k,u_k}\left(\beta_k(s_k) + \gamma_k(s_{k-1},s_k)\right). \tag{2. 24}$$

Both equations are add-compare-select (ACS) operations, which are similar to the path metric updating of Viterbi algorithm.

2.2.3 The Log-MAP algorithm

It can be figured out easily that Max-Log-MAP algorithm is a sub-optimal solution for turbo decoding since an approximation of (2. 21) is used to reduce the complexity of MAP algorithm. This problem can be solved by Log-MAP algorithm [11]. It employs the Jacobian algorithm

$$\begin{aligned}\log(e^{\delta_1}+e^{\delta_2}) &= \max(\delta_1,\delta_2)+\log(1+e^{-|\delta_1-\delta_2|}) \\ &= \max(\delta_1,\delta_2)+f_c(|\delta_1-\delta_2|),\end{aligned} \tag{2. 25}$$

where $f_c(|\delta_1-\delta_2|)$ is a correction function, and thus the performance can be improved. It has been proved that (2. 21) can be computed exactly by a recursive operation of (2. 25) [9].

$$\begin{aligned}\log(e^{\delta_1}+e^{\delta_2}+\cdots+e^{\delta_n}) &= \log(\Delta+e^{\delta_n}), \quad \Delta=e^{\delta_1}+e^{\delta_2}+\cdots+e^{\delta_{n-1}}=e^{\delta} \\ &= \max(\log\Delta,\delta_n)+f_c(|\log\Delta-\delta_n|) \\ &= \max(\delta,\delta_n)+f_c(|\delta-\delta_n|)\end{aligned} \tag{2. 26}$$

Substituting (2. 18) and (2. 19) into (2. 25), the forward and backward recursions can be represented as

$$\alpha_k(s_k)=\max_{s_{k-1},u_k}{}^{*}\left(\alpha_{k-1}(s_{k-1})+\gamma_k(s_{k-1},s_k)\right) \tag{2. 27}$$

and

$$\beta_{k-1}(s_{k-1})=\max_{s_k,u_k}{}^{*}\left(\beta_k(s_k)+\gamma_k(s_{k-1},s_k)\right), \tag{2. 28}$$

where the max*(.) operation is defined as

$$\max{}^{*}(\delta_1,\delta_2)=\max(\delta_1,\delta_2)+\log(1+e^{-|\delta_1-\delta_2|}). \tag{2. 29}$$

Finally, $L(\hat{u}_k)$ can be obtained by

$$\begin{aligned}L(\hat{u}_k) = &\max_{\substack{(s_{k-1},s_k)\\u_k=+1}}{}^{*}\left(\alpha_{k-1}(s_{k-1})+\gamma_k(s_{k-1},s_k)+\beta_k(s_k)\right) \\ &-\max_{\substack{(s_{k-1},s_k)\\u_k=-1}}{}^{*}\left(\alpha_{k-1}(s_{k-1})+\gamma_k(s_{k-1},s_k)+\beta_k(s_k)\right).\end{aligned} \tag{2. 30}$$

The performance of Log-MAP algorithm is identical to that of MAP algorithm. However, the complexity is also increased compared with Max-Log-MAP algorithm since computing f_c(.) still involves complicated exponentiations and multiplications. Thus, the values of f_c(.) are usually stored in a pre-computed table and Log-MAP algorithm can be implemented by table look-up. It has been found that excellent performance can be obtained with 8 stored values and $|\delta_1-\delta_2|$ ranging between 0 and 5, and no improvement is achieved with a finer representation [9].

2.2.4 SNR sensitivity of Max-Log-MAP and Log-MAP algorithm

Referring to (2.13) and its followed deductions, it’s evident that both MAP and log-MAP algorithm requires SNR estimation to obtain the value of channel reliability, i.e. L_c. Unfortunately, accurate estimation cannot be achieved easily. Several papers have discussed the effect of SNR mismatch in turbo decoding. In [12], the simulations show that about -3 to +6dB SNR estimation offset is tolerable before significant performance degradation. However, Max-Log-MAP algorithm is able to provide a SNR independent scheme if *a priori* information is initialized with a reasonable value, such as all zeros for each state [13]. Due to the linearity of max(.) operations, the term L_c can be canceled out while computing $L(\hat{u}_k)$. The comparison of Max-Log-MAP and Log-MAP algorithm under different SNR estimation offsets was made in [13].

Although Log-MAP algorithm provides the performance better than that of Max-Log-MAP algorithm, it suffers the risk of serious SNR mismatch offset. Thus, channel characteristics play an important role in practical implementation. It has been concluded in [13] that if channel characteristics change over time, the Max-Log-MAP decoder is suitable to be the constituent decoder in turbo decoding. Otherwise, Log-MAP decoder should be preferable in the aspect of coding gain.

2.3 *Sliding Windowed Approach*

As what we described in the previous section, the MAP-series algorithm (including MAP algorithm, Max-Log-MAP algorithm, and Log-MAP algorithm) requires the entire block message to be received before decoding procedure can be started since backward path metric computation needs information at the end of trellis. This restriction enlarges the memory requirement for hardware implementation of turbo decoder. For example, the maximum block length of 3GPP2 standard is 20730, which means 20730 metrics should be stored. Besides, long output latency is also introduced. This is disadvantageous for turbo code in real-time application.

A simple method to solve these problems is to divide data stream into many sub-blocks. However, the last bits in these sub-blocks suffer lower error tolerance because of the lack of initial metrics for backward recursion. Thus, a sliding windowed approach was proposed in [14] and later on in [15] to overcome this drawback. It utilizes the fact that the backward path metrics can be highly reliable even without knowing the initial state if the backward recursion goes long enough. The windowed processing schedule used in our design is illustrated in Fig. 2.7 and the detail operating flow is described as follows.

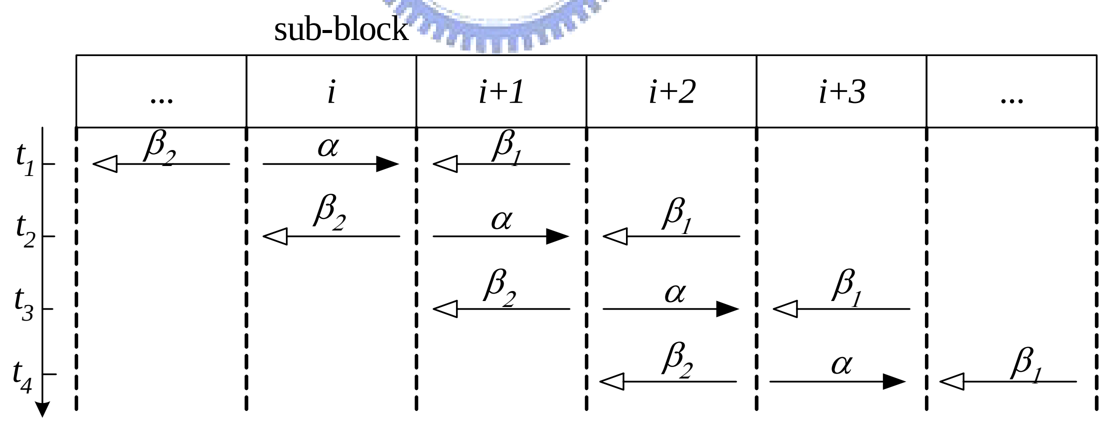


Fig. 2.7 The windowed MAP algorithm

Initially, the received data block is divided into many sub-blocks, with a sub-block length of *L*. *L* is called the convergence length. Typically, it's about five times the constraint length of the encoder. In 3GPP2 standard, the constraint length is 4. For each sub-block *i*, the forward recursion computes the forward path metrics α and storing these values into memory. In parallel, an additional backward recursion β_1 is performed in the next sub-block i+1. Once β_1 operation in sub-block i+1 is finished, the last backward path metric obtained for each state is regarded as a reliable initial β for sub-block i to start its backward recursion, which is labeled as β_2 in Fig. 2.7. Finally, the $L(\hat{u}_k)$ can be computed by α, β_2, and γ. Fig. 2.8 shows the influence of different sub-block lengths on the performance of turbo code.

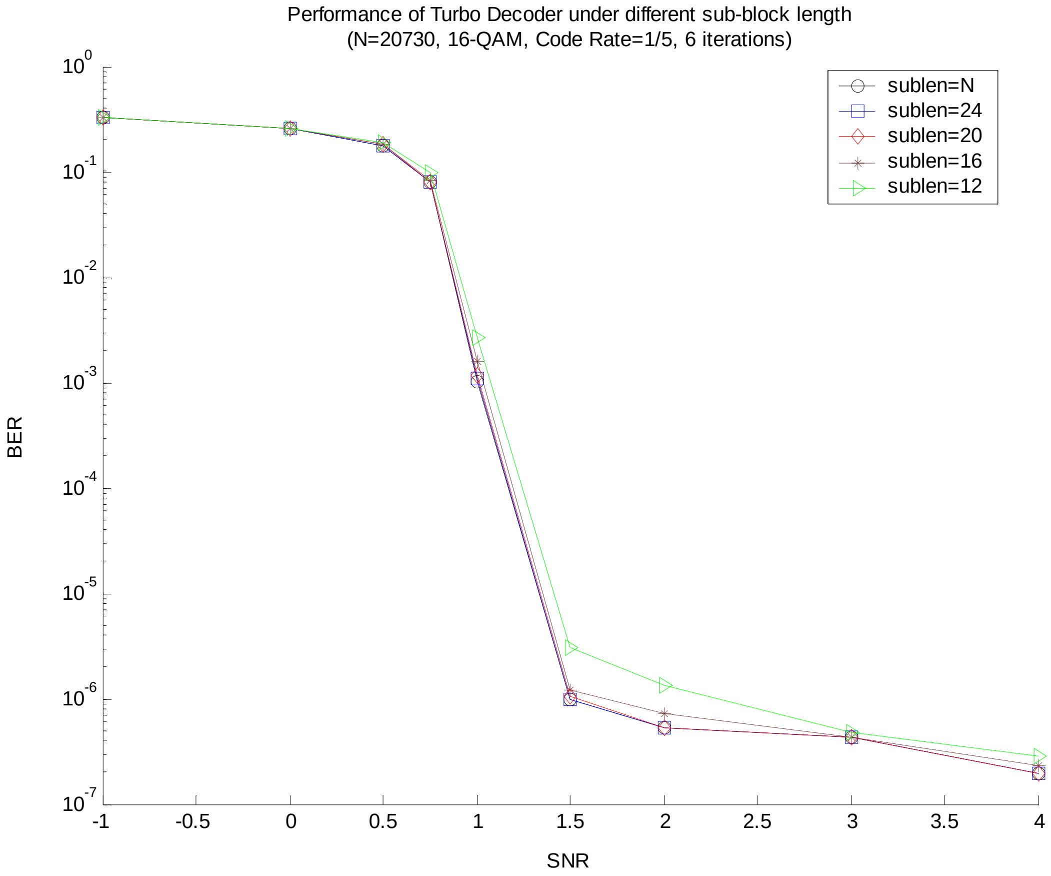


Fig. 2.8: Performance comparison among different sub-block lengths in 3GPP2 standard

Chapter 3

Principle of Convolutional codec

3.1 Convolutional Code

For the convolutional code, its encoder is constructed with several memory elements and modulo-2 adders. In general, it is usually expressed as a (*n*, *k*, *v*) convolutional encoder where *n*, *k*, *v* are the number of output, the number of input and the number of memory elements respectively. 3GPP2 standard specifies rate 1/2, 1/3, 1/4, and 1/6 convolutional codes. All of them have a constraint length of 9. An example of rate 1/2 convolutional encoder with the generator matrix of $[753, 561]_{(octal)}$ is illustrated in Fig. 3.1. For each input information bit, it generates two code symbols (c_0 and c_1) by the generator matrix. The generator matrices for other code rate convolutional codes are listed in Table 1.2. The convolutional encoder should be initialized with all-zero state.

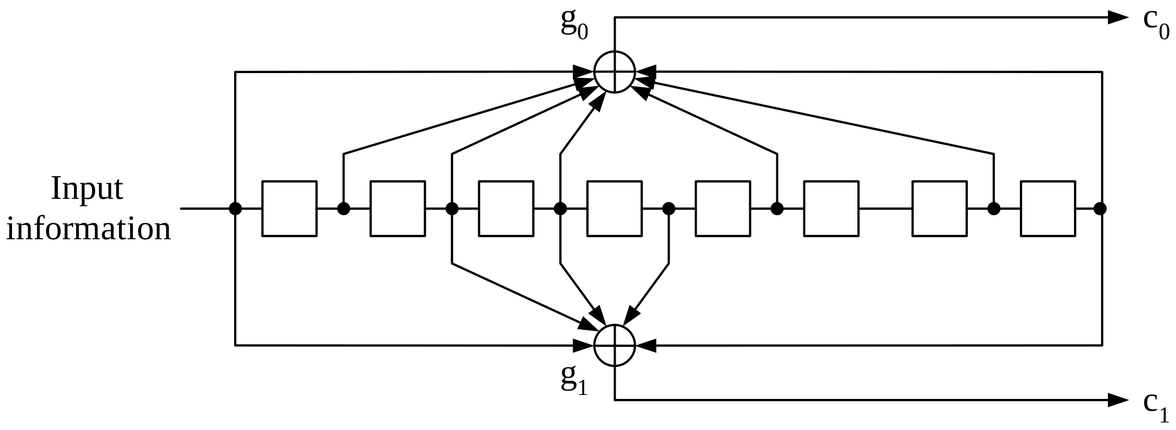


Fig. 3.1 Rate 1/2 convolutional encoder with the constraint length of 9

3.2 *Viterbi Decoding*

Up to now, Viterbi algorithm [16] is the optimal solution to decode the convolutional code. It utilizes the maximum likelihood decoding algorithm and searches the shortest path through a weighted graph. In fact, Viterbi algorithm has become a standard due to its fairly decoding complexity. Before explaining Viterbi algorithm, a system platform, which is shown in Fig. 3.2, should be interpreted first.

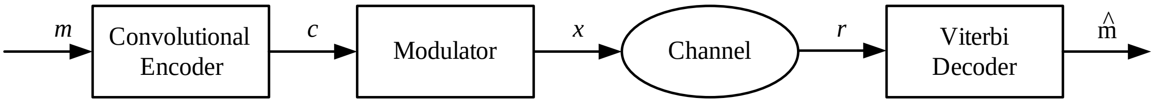


Fig. 3.2 A system platform of the convolutional codec

Initially, the message sequence m is encoded into the codeword sequence c. After signal modulation, the modulated sequence x is transmitted into the channel. In the receiver, the sequence $\boldsymbol{r}$ is received. The major concept of Viterbi algorithm is to find the maximum likelihood sequence $\hat{m}$ according to $\boldsymbol{r}$. Theoretically, it's equivalent to maximize the probability of $P(m|\boldsymbol{r})$. Using Baye's rule

$$P(m\,|\,\boldsymbol{r}) = \frac{P(m)\cdot P(\boldsymbol{r}\,|\,m)}{P(\boldsymbol{r})} \tag{3. 1}$$

where $P(\boldsymbol{r})$ is independent of m. Thus, what the decoder does is to maximize the probability of $P(\boldsymbol{r}\mid m)$. Assume the length of the received sequence is τ; then $P(\boldsymbol{r}\mid m)$ can be expressed as

$$\begin{aligned} P(\boldsymbol{r}|m) &= P(\boldsymbol{r}\,|\,x) \\ &= \prod_{t=1}^{\tau} P(\boldsymbol{r}_t\,|\,x_t) \\ &= \prod_{t=1}^{\tau}\prod_{i=0}^{n-1} \frac{1}{\sqrt{2\pi\sigma}} \exp\left(-\frac{(r_{t,i}-x_{t,i})^2}{2\sigma^2}\right) \end{aligned} \tag{3. 2}$$

Similarly, these works can be transformed to logarithm domain to reduce computing complexity. The probability $P(\boldsymbol{r}|m)$ in logarithm domain is given by

$$\log P(\boldsymbol{r} \mid m) = \sum_{t=1}^{\tau} \log P(\boldsymbol{r}_t \mid x_t) \tag{3. 3}$$

For AWGN channel, (3. 3) can be further rewritten as

$$\begin{aligned} \log P(\boldsymbol{r} | m) &= \sum_{t=1}^{\tau} \log \prod_{i=0}^{n-1} \frac{1}{\sqrt{2\pi\sigma}} \exp\left(-\frac{(\boldsymbol{r}_{t,i} - x_{t,i})^2}{2\sigma^2} \right) \\ &= -\frac{n\tau}{2} \log(2\pi) - n\tau \log(\sigma) - \sum_{t=1}^{\tau} \sum_{i=0}^{n-1} \frac{(r_{t,i} - x_{t,i})^2}{2\sigma^2} \end{aligned} \tag{3. 4}$$

In other words, to maximize the probability of $P(\boldsymbol{r} \mid m)$ is to minimize Euclidean distance.

$$\sum_{t=1}^{\tau} \sum_{i=0}^{n-1} \frac{(r_{t,i} - x_{t,i})^2}{2\sigma^2} \tag{3. 5}$$

In order to compute Euclidean distance, Viterbi algorithm defines the branch metric (BM, also called transition metric, or simply TM) $\lambda_t(s_{t-1}, s_t)$ and the path metric Γ_{t,s_k} as follows.

$$\lambda_t(s_{t-1}, s_t) = \sum_{i=0}^{n-1} \frac{(r_{t,i} - x_{t,i})^2}{2\sigma^2} \bigg|_{S_{t-1}=0,1,\ldots,2^{\nu}} \tag{3. 6}$$

$$\Gamma_{t,s_k} = \min\left(\Gamma_{t-1,s_{t-1}} + \lambda_t(s_{t-1}, s_t)\right) \tag{3. 7}$$

It is clear that the path metric Γ_{t,s_k} is the minimum Euclidean distance for state s_t. At each time index, the decoder computes and compares the metrics of all branches that entering the state. The branch with the minimum metric and its corresponding decision bit will be preserved and others will be eliminated. The history record of the decision bits is called survivor. According to the minimum path metric at each time index, the maximum likelihood sequence can be estimated.

Finally, the steps of Viterbi algorithm can be summarized as follows.

1. Initialize all path metric storages and survivor memory.
2. According to the received sequence ***r***, compute the branch metric $\lambda_t(s_{t-1}, s_t)$ for each state transition.
3. Accumulate the path metric with the branch metric that will converge toward the same

state.

4. Update the path metric storage for each state according to the following principle.

$$\Gamma_{t,s_k} = \min\left(\Gamma_{t-1,s_{t-1}} + \lambda_t(s_{t-1}, s_t)\right)$$

The decision bit of each state is also stored into survivor memory at the same time.

5. Decode the message sequence according to the minimum path metric and the survivor.
6. Repeat this process until all messages are decoded.

3.3 *Trace-back Method*

The trace-back method is a technique to trace the maximum likelihood sequence in the survivor memory. Here we'll use a (2, 1, 2) convolutional code with the generator matrix of $[111, 101]_{\text{binary}}$ as the example. Its trellis diagram and the corresponding contents of the survivor memory are shown in Fig. 3.3. In this figure, all dotted lines represent the eliminated paths. Once the upper path entering into this state is chosen, the decision bit is set as zero; otherwise, it'll be set to one.

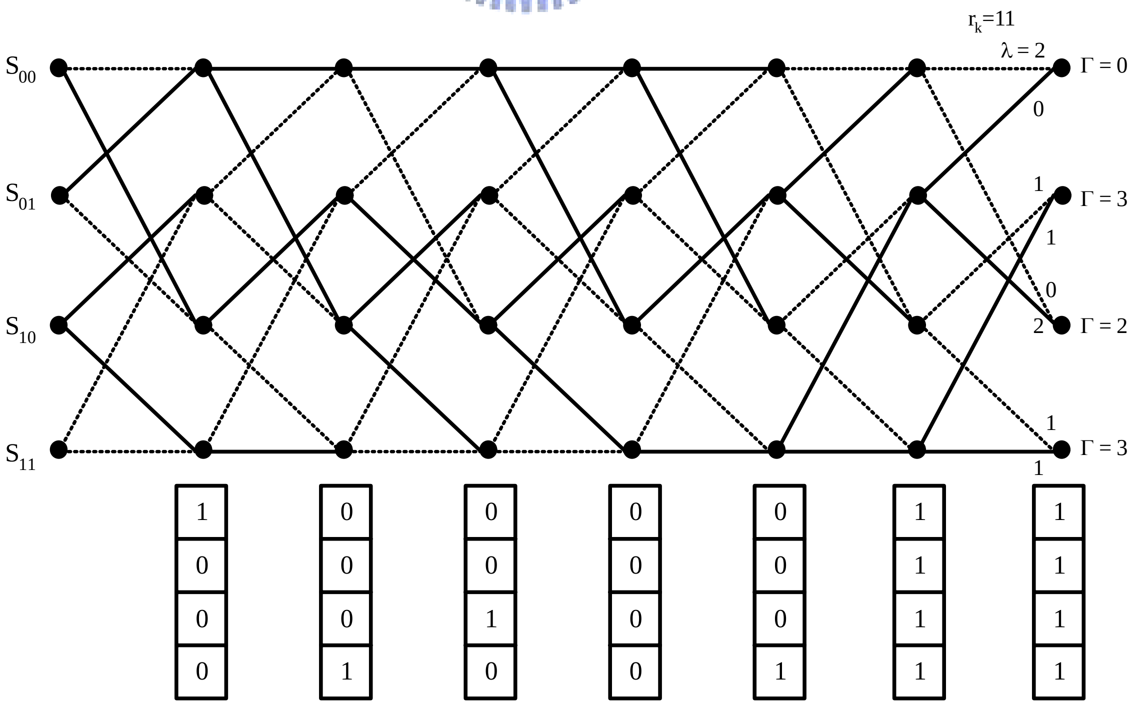


Fig. 3.3 Trellis diagram of the (2, 1, 2) convolutional code and its survivor memory

After all symbols are received, the maximum likelihood sequence can be decided by trace-back method. This procedure starts from the state with minimum path metric happened in S_{00}. By recursively shifting the state number left and inserting the decision bit stored in the survivor memory back to the right hand side, decoding procedure can be completed. The overall trace-back operation of the example in Fig. 3.3 is illustrated with Fig. 3.4.

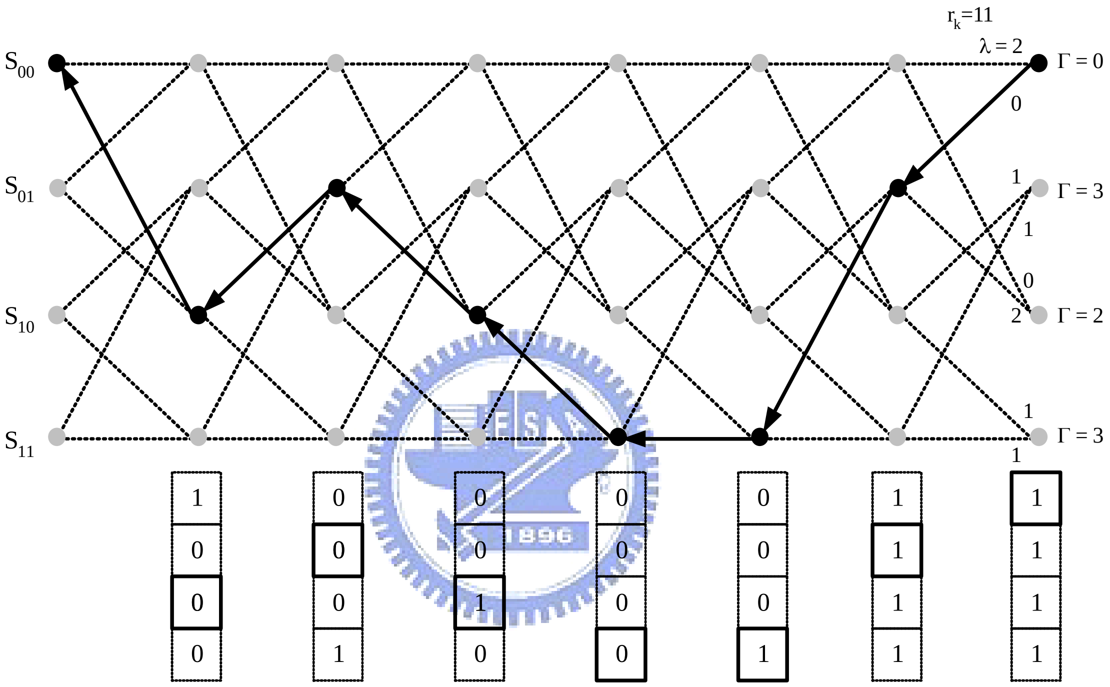


Fig. 3.4 Trace-back procedure of the convolutional code

In fact, the length of received symbols may be quite long. If we don’t start the trace-back operation until all symbols are received, an extremely large survivor memory is required. This is impossible for chip realization. Thus, a suitable trace-back length should be defined without serious performance degradation. Similar to what we introduced in section 2.3, it’s about five times the constraint length of the encoder. A simulation result under different trace-back lengths is shown in Fig. 3.5.

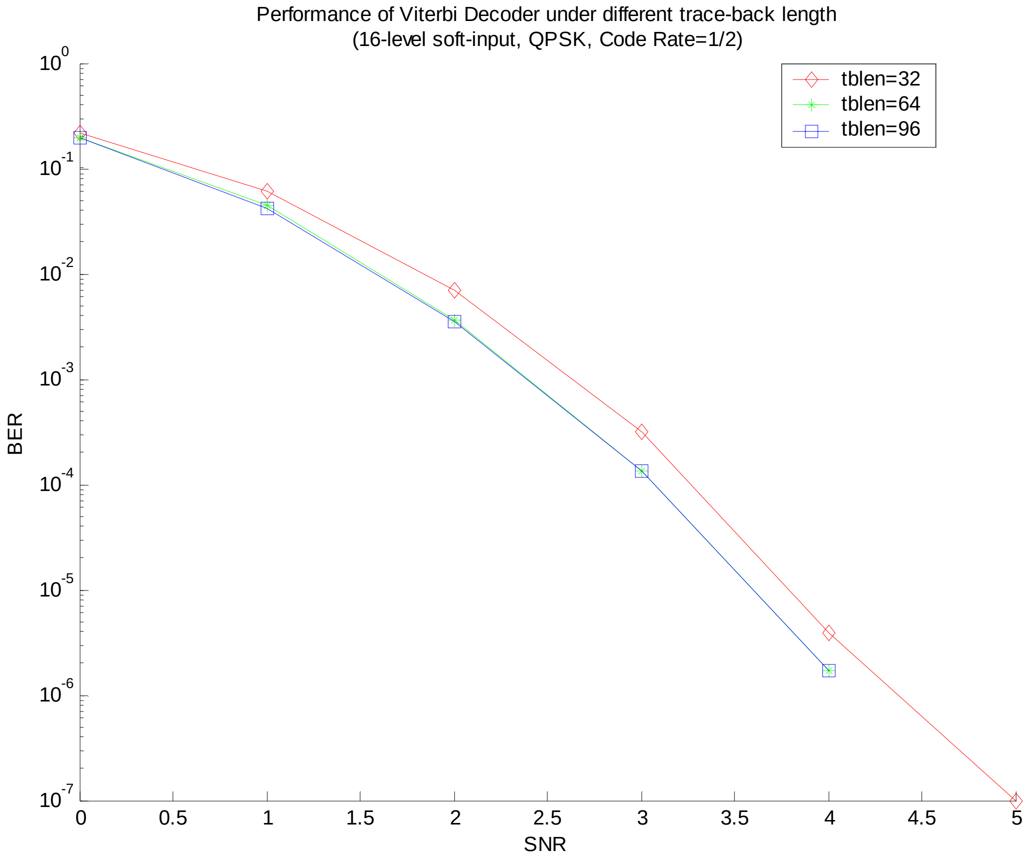


Fig. 3.5 The simulation result of Viterbi decoder under different trace-back lengths

3.4 Summary

In this chapter, we have made some brief descriptions about convolutional code. Viterbi algorithm, which is widely applied in convolutional decoding, is also presented here. Due to power-saving issue, instead of using register-exchange method, we use trace-back method to manage the storage of the survivor sequences. A suitable track-back length is chosen according to the simulation result shown in Fig. 3.5. This will help us reduce the memory requirement with little performance degradation.

Chapter 4

Fixed Point Analysis of Dual Mode Turbo/Viterbi Decoder

Turbo decoding uses SISO decoders to achieve a fairly good coding performance. In previous chapter, we analyze several factors that affect the characterization of turbo code by importing floating point information as the required soft-input. However, all soft-inputs should be bounded since infinite precision is impossible to be achieved for the practical implementation. In general, coding performance may suffer quantization loss due to internal bit-width limitation. A trade-off between hardware cost and the performance must be concerned before the chip implementation. For turbo decoder, the Max-Log-MAP decoder is adopted as our SISO decoder because of the lack of the channel characteristics in our 3GPP2 simulation platform. Thus, the following fixed point analysis will base on the Max-Log-MAP algorithm. For Viterbi decoder, a sixteen level soft-input precision is proposed. The relative bit-width for path metric computation is also discussed here. In this chapter, we'll show an optimal solution that the hardware cost can be minimized without critical performance degradation for turbo/Viterbi decoder in 3GPP2 standard.

4.1 Fixed Point Analysis for Turbo Decoder

4.1.1 Input LLR Representation

Quantization of input LLR will directly influence not only the performance of Turbo decoding but also the memory requirement of the design. The reason is obvious that since the received systematic symbols should be interleaved for the second SISO decoder, the memory size will be directly proportional to the bit-width of systematic symbol. To determine the acceptable finite precision of the input LLR, several simulations under AWGN channel with BPSK and 16-QAM have been performed using a floating point turbo decoder with quantized input LLRs. Note that the value of channel reliability "L_c" can be assumed as 1 for the practical implementation without performance degradation. Hence, the received symbol vector introduced in section 2.2 can be regarded as input LLR directly. Fig. 4.1 plots the quantization loss of the bounded input symbols with BPSK modulation and rate 1/2 turbo decoding. Fig. 4.2 shows the same simulation but with 16-QAM modulation and rate 1/5 turbo decoding. The former case is used while data and signaling are transmitted from a mobile station to a base station; and the later case is used while data are sent in the opposite direction. The dotted line is a rough threshold corresponding to 5% frame error rate (FER), which is the target FER specified in 3GPP2 standard. Note that $a.b$ shown in these figures denotes the quantization scheme where a is the number of bits used for the integer part, and b is the number of bits used for the fractional part. Simulation result shows that the performance of 3.3 scheme is slightly worse than that of 4.2 scheme in Fig. 4.2. Nevertheless, the performance is going to be better than others in Fig. 4.1, and thus is recommended for the Max-Log-MAP decoder.

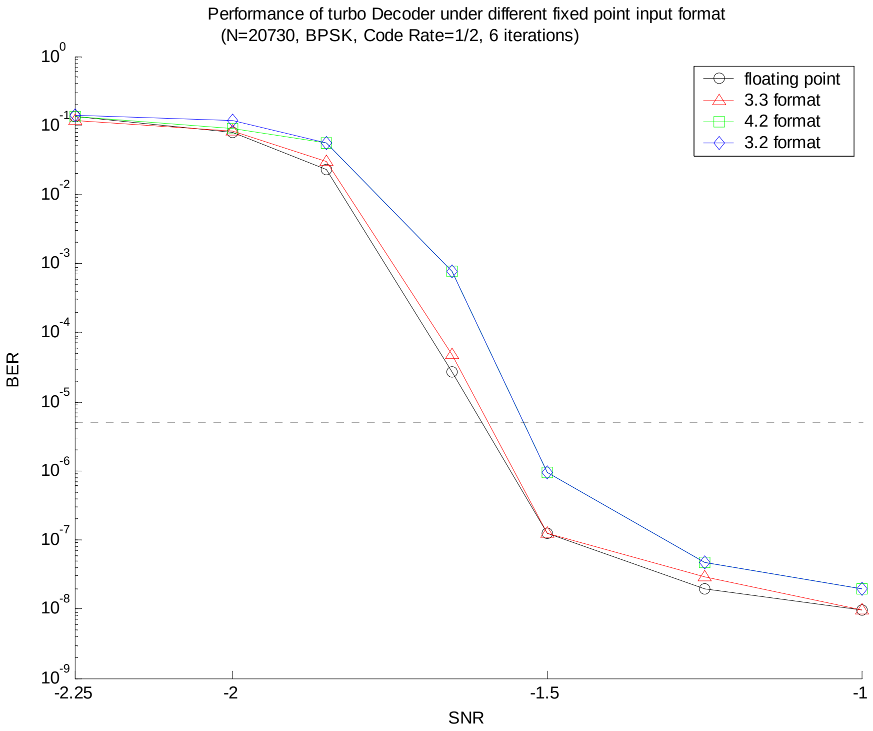


Fig. 4.1 Fixed point simulation result of the input symbols with BPSK modulation and rate 1/2 turbo decoding

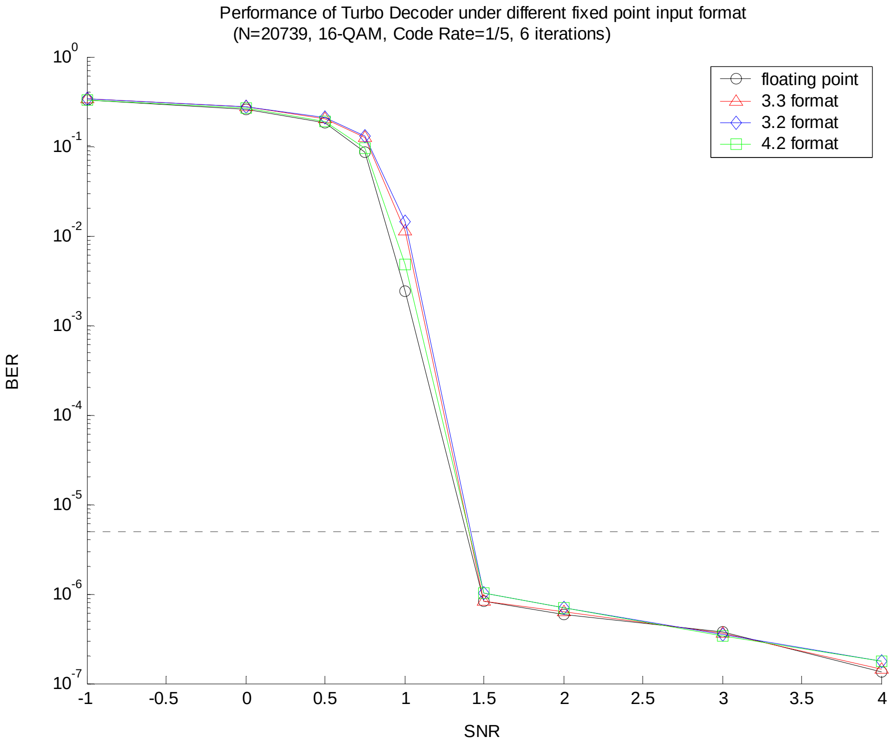


Fig. 4.2 Fixed point simulation result of the input symbols with 16-QAM modulation and rate 1/5 turbo decoding

4.1.2 Extrinsic Data Representation

Extrinsic data provides important information for turbo codes to perform the iterative decoding. Quantization of extrinsic information should be done carefully or the effect of iterative decoding will be lessened. Strictly to say, 8-bit integer is necessary for our proposed design to avoid overflow, which will be explained later in section 4.1.3. However, this is a heavy burden in hardware implementation since it should be interleaved or de-interleaved to be *a priori* information of the other SISO decoder for turbo decoding, and thus large memory is required. Thus, a fixed point simulation of the extrinsic data for cost consideration is performed, as shown in Fig. 4.3. Note that any extrinsic information exceeding the range that can be expressed is pulled back the nearest value instead of truncating it directly. This will ensure that no meaningless performance degradation occurs. The idea comes from the fact that the extrinsic data is averagely small while channel noise is averagely high. On the contrary, it will be large if channel provides good transmission quality. In the former case, the data value is small so that bit-width requirement can be surely truncated. In the later case, since the transmission quality is good, the extrinsic information just needs to be large enough with little influence on the error correction ability of iterative decoding. Therefore, the bit-width can be reduced for both cases. We can see that with four or more extrinsic data integer bits, the performance is close to that of floating point simulation result. So, the 4.2 scheme should be the best choice.

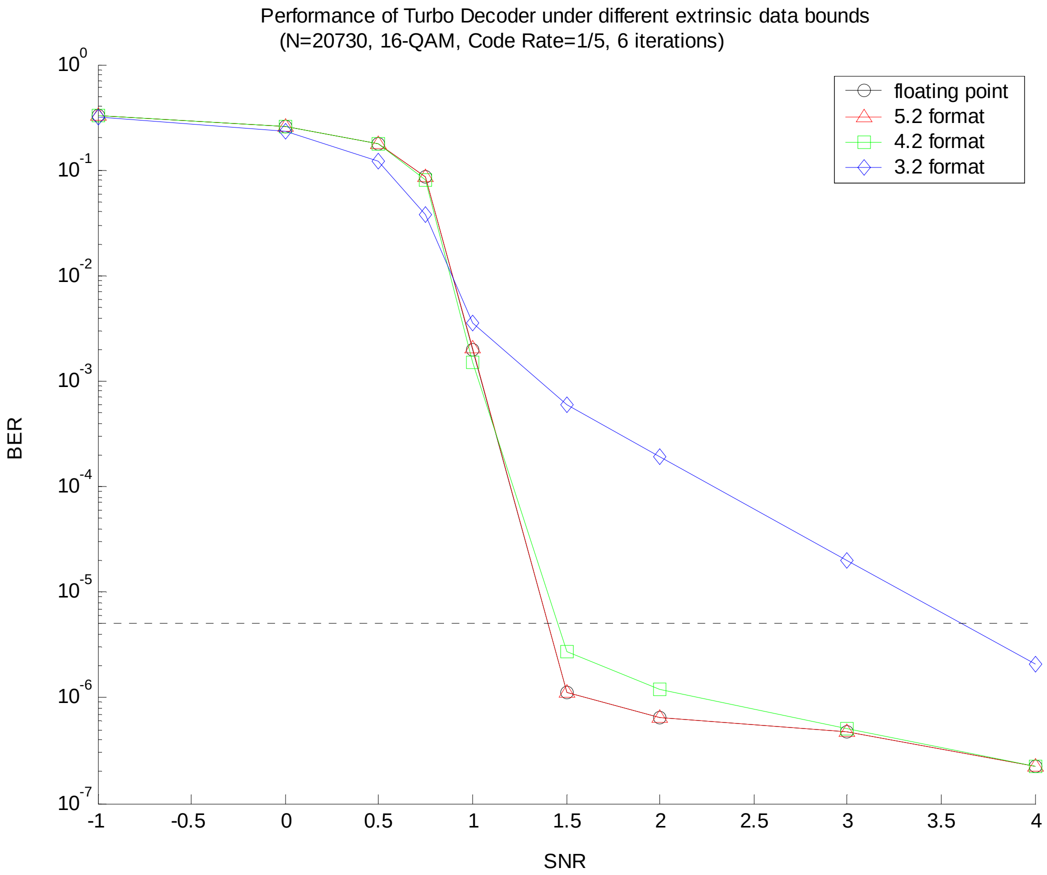


Fig. 4.3 Fixed point simulation result of the extrinsic information

4.1.3 Bit width of Internal Variables

In previous sections, we had determined the range of all input information, including input LLRs and *a priori* data that comes from the extrinsic data of the other SISO decoder. According to the bounded inputs, the bit width of internal variables, γ_k, α_k, and β_k, for Max-Log-MAP decoding can be derived. Firstly, the bound of γ_k can be decided by (2.14) and (2.17). Given integer range of B_{in} for input LLRs and that of B_{Lex} for *a priori* data, the maximum difference of γ_k is denoted as $\Delta\gamma_k$ and derived by

$$\Delta\gamma_k \leq n \times B_{in} + B_{Lex} \tag{4. 1}$$

In 3GPP2 standard, the code rate of component encoder is 1/3 and thus the value "*n*" in (2.14) should be 3. For B_{in}=8 (3-bit integer) and B_{Lex}=16 (4-bit integer), the maximum difference of γ_k in our design should be 40, and thus 6-bit integer are required.

In spite of the recursion of α_k and β_k computation, the range of the differences between path metrics is still bounded and provided with the following theorem by [17] and [18].

Theorem: For an RSC encoder with m shift-registers and a maximum Hamming distance of d_m between any two paths across m trellis sections, the difference of path metrics is bounded by

$$\Delta\alpha_k \le m \times B_{Lex} + d_m \times B_{in} \tag{4. 2}$$

and

$$\Delta\beta_k \le m \times B_{Lex} + d_m \times B_{in} \tag{4. 3}$$

The theorem utilizes the fact that the probabilities of any two states at time index k originate from the same set of states at time index k-m. Thus, the difference of any two path metrics at time index k is dependent only on the branch metrics from time index k-m to k. Fig. 4.4 illustrates the paths of metrics passing with m=3. Let $S_{\max}$ be the state with maximum path metric at time index k, and $S_{\min}$ be that with minimum path metric at time index k. Then, the bound can be expressed as

$$\alpha(S_{\max}) - \alpha(S_{\min}) = M_{\max} - M_{\min} \tag{4. 4}$$

where $M_{\max}$ and $M_{\min}$ are the accumulated branch metrics from time index k-m to k for $S_{\max}$ and $S_{\min}$ respectively. The result tells us clearly that the bound is determined by the maximum difference of branch metrics within m trellis sections. For 3GPP2 standard, these two paths are labeled in Fig. 4.4. For B_{in}=8 and B_{Lex}=16, the corresponding bound is 96 and its relative bit-width for integer part is 7.

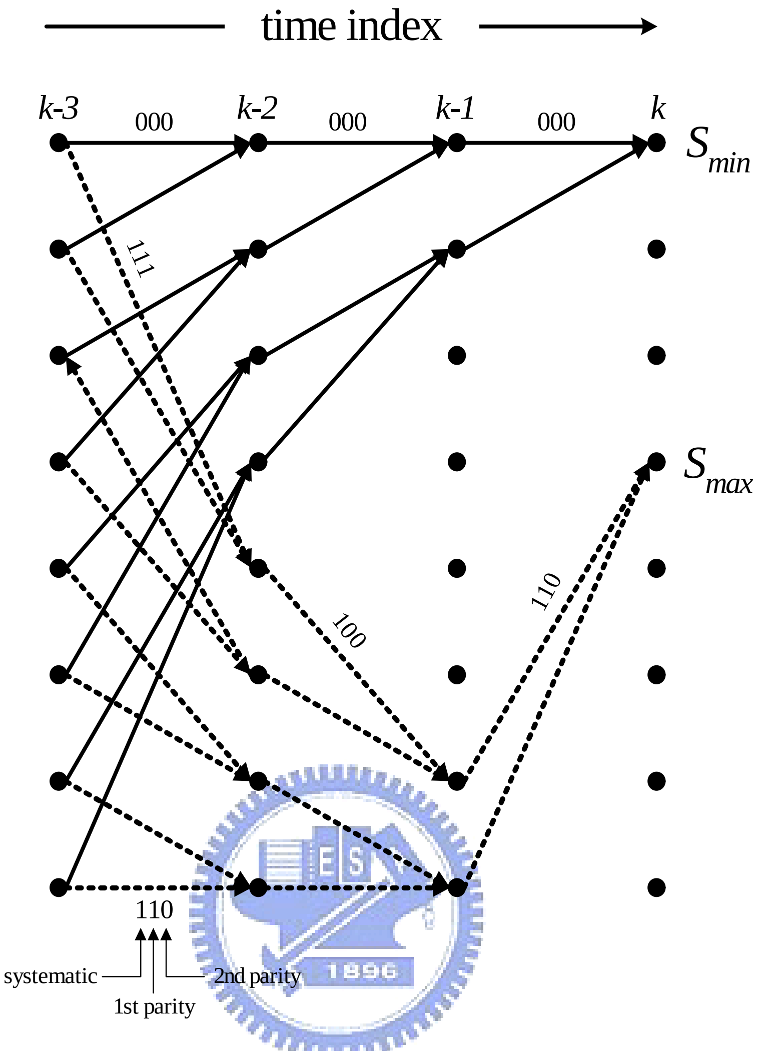


Fig. 4.4 An eight-state trellis diagram illustrating message passing within 3 trellis sections.

After evaluating the bounds of γ_k, α_k, and β_k, the bound on the magnitude of the output LLR, $L(\hat{u}_k)$, can be derived with the following theorem by [17] and [18].

Theorem: Given $\Delta\alpha_k$ as the bound of the difference of the forward path metric, and $\Delta\gamma_k$ as the bound of the difference of the branch metric, the magnitude of the output LLR is bounded by

$$|L(\hat{u}_k)| \le \Delta\alpha_k + \Delta\gamma_k \tag{4. 5}$$

The theorem is simply derived from the definition of $L(\hat{u}_k)$ listed in (2.15). Replace $\alpha_{k\text{-}1}(s_{k\text{-}1})$ and $\gamma_k(s_{k\text{-}1},s_k)$ in the numerator of (2.15) by $max(\alpha_{k\text{-}1})$ and $max(\gamma_k)$, respectively. Also replace $\alpha_{k\text{-}1}(s_{k\text{-}1})$ and $\gamma_k(s_{k\text{-}1},s_k)$ in the denominator of (2.15) by $min(\alpha_{k\text{-}1})$ and $min(\gamma_k)$, respectively. Then (2.15) can be extended to

$$L(\hat{u}_k) \le \ln \frac{\max \alpha_{k-1}(S_{k-1}) \cdot \max \gamma_k(S_{k-1}, S_k) \cdot \sum\limits_{S_k, u_k=+1} \beta_k(S_k)}{\min \alpha_{k-1}(S_{k-1}) \cdot \min \gamma_k(S_{k-1}, S_k) \cdot \sum\limits_{S_k, u_k=-1} \beta_k(S_k)} \tag{4. 6}$$

Since each state at time index k originates from two branches, exactly one of which is corresponding to an information bit of 0 and the other one must be corresponding to an information bit of 1, we can obtain

$$\sum_{S_k, u_k=+1} \beta_k(S_k) = \sum_{S_k, u_k=-1} \beta_k(S_k) \tag{4. 7}$$

and (4. 6) can be further simplified as

$$L(\hat{u}_k) \le \Delta\alpha_{k-1} + \Delta\gamma_k \tag{4. 8}$$

Similarly, by replacing $\alpha_{k\text{-}1}(s_{k\text{-}1})$ and $\gamma_k(s_{k\text{-}1},s_k)$ in the numerator of (2.15) by $min(\alpha_{k\text{-}1})$ and $min(\gamma_k)$, respectively, and those in the denominator of (2.15) by $max(\alpha_{k\text{-}1})$ and $max(\gamma_k)$, respectively, the lower bound of $L(\hat{u}_k)$ can be obtained by

$$L(\hat{u}_k) \ge -\Delta\alpha_{k-1} - \Delta\gamma_k \tag{4. 9}$$

Theoretically, the range of $L(\hat{u}_k)$ in our proposed design should be between -136 and +136. However, the probability of $|L(\hat{u}_k)| \ge 128$ is extremely small. For cost consideration, we can use only 8 bits to represent the integer part of $L(\hat{u}_k)$ with little performance degradation.

Finally, the bound on the magnitude of the extrinsic information $L_{ex}(u_k)$ can be derived base on (2. 15) and formulated as follows.

$$|L_{ex}(u_k)| \le |L(\hat{u}_k) - L_c r_{k,1} - L(u_k)| \tag{4. 10}$$

In our case, the corresponding bound should be $|L_{ex}(u_k)| \le 124$. Hence, the suitable bit-width for $L_{ex}(u_k)$ is the same as that of $L(\hat{u}_k)$.

4.1.4 Performance under Fixed Point Simulation

After confirming the bound of each internal variable, a simulation for cost-down of path metric storage is performed in Fig. 4.5. For the 6.2 scheme, the performance is matched to that of 7.2 scheme, which is the upper bound of the path metric. This result is not strange. The drastic condition discussed in section 4.1.3 will occur unless all received input symbols, including *a priori* probability, simultaneously reach the relative large values cross three time sections in trellis successively. Such event is seldom observed. Consequently, the 6.2 scheme is chosen in our design. Finally, we simulate the performance of whole turbo decoder under complete fixed point condition. In Fig. 4.6, it shows that there is only about 0.25 dB design loss compared with the floating point simulation result in AWGN channel.

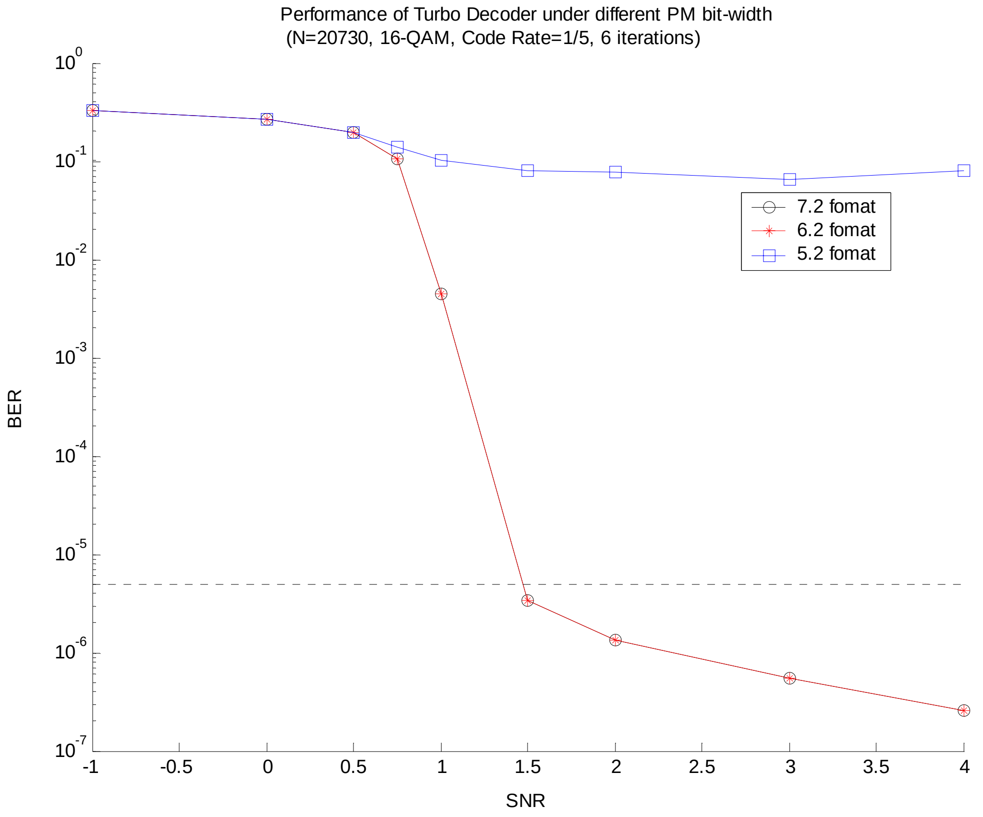
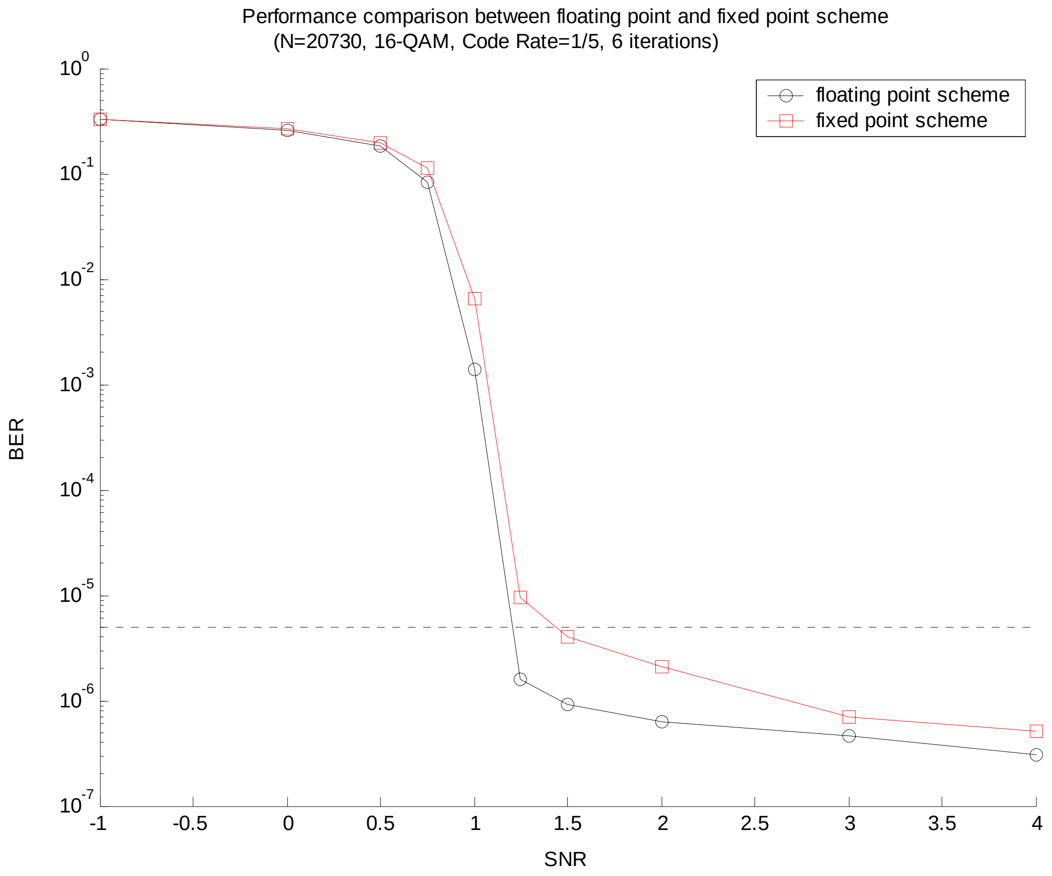


Fig. 4.5 Fixed point analysis with different bit-width of path metrics in turbo decoding



1896

Fig. 4.6 Design loss of fixed point turbo decoder

4.2 *Fixed Point Analysis of Viterbi Decoder*

Soft decision Viterbi decoder provides a better error correction capability. With increasing quantization level, the error probability can be further reduced with a penalty of linearly increased complexity. However, the degree of improvement will saturate as the quantization level reaches a threshold. A simulation to evaluate the performance improvement with different quantization level is done and shown in Fig. 4.7. All schemes are set to be uniform quantization and optimal step size. The BER curve with 128-level soft-input is assumed to be the performance limitation of code rate 1/2 256-state Viterbi decoding. As we can see, the improvement from the scheme with 8-level soft-input to that with 16-level is up to 0.4dB. Nevertheless, the 32-level scheme gains about only 0.2dB from 16-level scheme.

Hence we can conclude that the 16-level soft decision yields a good trade-off between performance and complexity and thus is chosen for the proposed design.

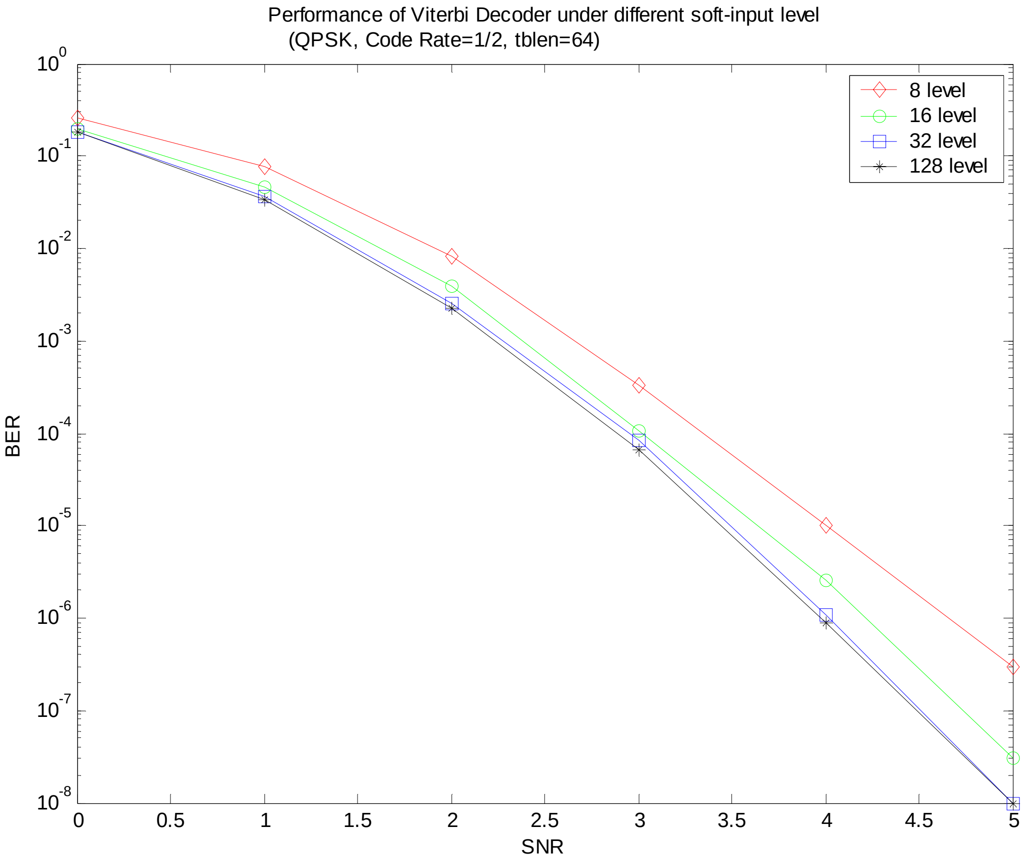


Fig. 4.7 The performance of Viterbi decoder with different quantization levels

To restrict the difference of any two states at any time is also a critical issue for Viterbi decoding. According to Viterbi algorithm, it supposes that all survivor paths will converge to the same node among the truncation length *L*. This assumption can be expressed by Fig. 4.8. A principle of choosing the truncation length is introduced in section 3.3. Computing all path metrics from *t*=0, we can get

$$\Gamma_k(S_1) = \Gamma_1 + \Gamma_{k-L} \tag{4. 11}$$

$$\Gamma_k(S_2) = \Gamma_2 + \Gamma_{k-L} \tag{4. 12}$$

Then, the difference of any two path metrics can be written as

$$|\Gamma_k(S_1) - \Gamma_k(S_2)| = |\Gamma_1 - \Gamma_2| \leq BL \qquad (4.\ 13)$$

where B denotes the maximum value of the branch metric.

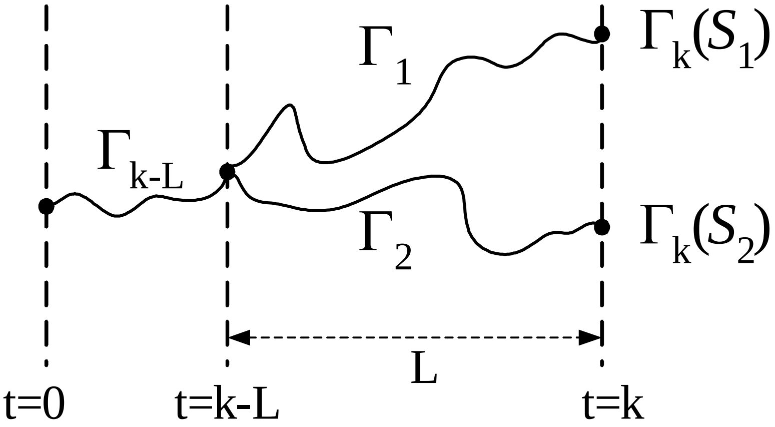


Fig. 4.8 The convergence of any two survivor paths in Viterbi algorithm

In 3GPP2 standard, the minimum code rate is 1/6. Combining with 16-level soft input, the value of B is 90. Therefore, the upper bound of path metric in our proposed design should be 5760, which means 13 bits at most are required theoretically. However, this case rarely happens. In fact, the bit-width of path metric will directly influence the size of its storage. A simulation for the cost consideration of path metric storage was done and the result is shown in Fig. 4.9. It indicates that obvious performance degradation occurs while 9-bit scheme is adopted. Therefore, we'll use 10 bits for path metric representation in Viterbi decoder, and corresponding storage requirement should be at least 2560 bits. Finally, a performance analysis on system performance for each supported code rate is concluded in Fig. 4.10.

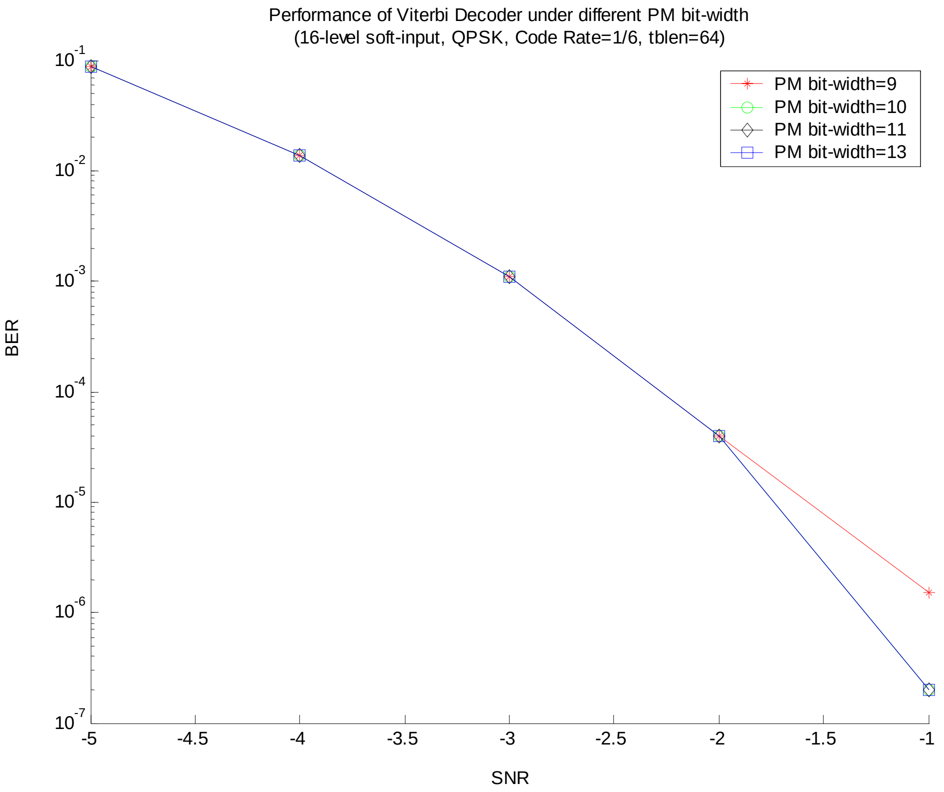


Fig. 4.9 The performance of Viterbi decoder under different bit-widths of path metric

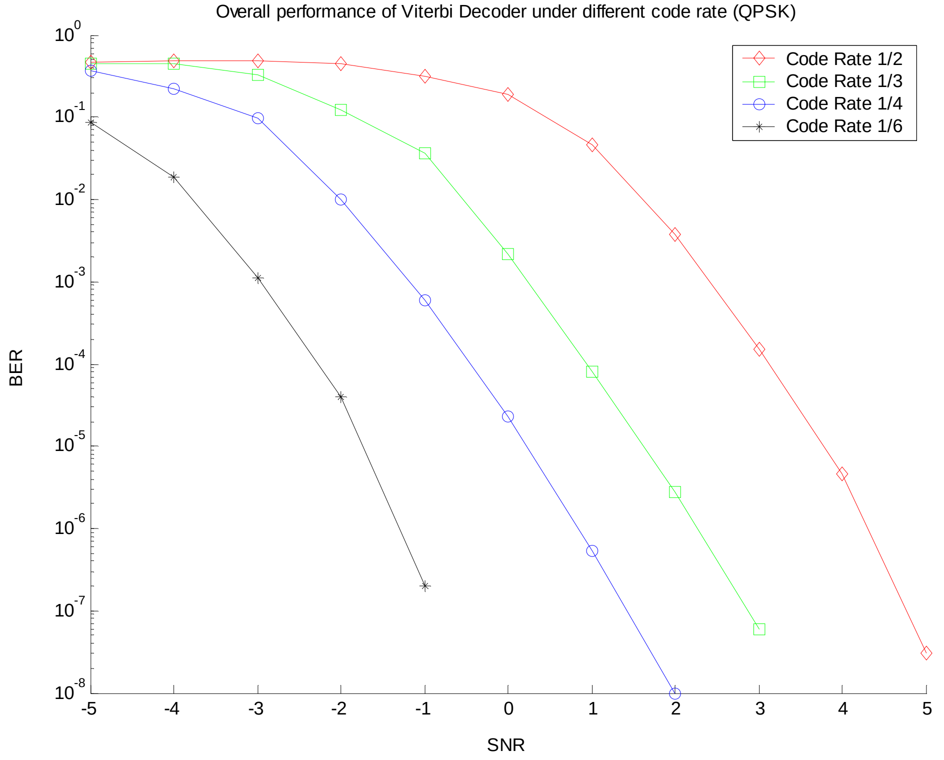


Fig. 4.10 The performance analysis on system performance for each kind of code rate

4.3 Summary

In this chapter, some fixed point performance analysis for internal variables is done for both operating modes. Summaries of bit-width decision for turbo mode and Viterbi mode are made in Table 4.1 and Table 4.2 respectively. It is verified that the design loss is only about 0.25dB for both turbo mode and Viterbi mode. A comparison with other similar work [19] for turbo mode is also made in Table 4.3. It shows that the results are nearly the same.

Table 4.1 Summary of bit-width decision for turbo mode

Variables	$L_c r_{k,v}$	$L(u_k)$	$\Delta\gamma$	$\Delta\alpha$	$\Delta\beta$	$L(\hat{u}_k)$	$L_{ex}(u_k)$
Bounds	$-\infty \sim +\infty$	-8 ~ +8	40	96	96	-136 ~ 136	-124 ~ 124
Bit-width	6 (3.3)	6 (4.2)	8 (6.2)	8 (6.2)	8 (6.2)	9 (7.2)	9 (7.2)

Table 4.2 Summary of bit-width decision for Viterbi mode

Variables	soft input	ΔBM	ΔPM
Bounds	0 ~ 15	0 ~ 90	0 ~ 5760
Bit-width	4	7	10

Table 4.3 A comparison of bit-width decision with [19] for turbo mode

Variables	$L_c r_{k,v}$	$L(u_k)$	$\Delta\gamma$	$\Delta\alpha$	$\Delta\beta$	$L(\hat{u}_k)$	$L_{ex}(u_k)$
our work	6 (3.3)	6 (4.2)	8 (6.2)	8 (6.2)	8 (6.2)	9 (7.2)	9 (7.2)
[19]	5 (3.2)	6 (4.2)	5*	6*	6*	8*	8*

* required bits for integer part only.

Chapter 5

Architecture of Proposed Dual Mode Turbo/Viterbi Decoder

5.1 Architecture of Integrated Turbo/Viterbi Decoder

Because of the trellis decoding structure of both decoders, the combination takes the advantage of resource sharing in the ACS and memory unit, leading to a much compact architecture for 3GPP2 system. The proposed architecture of integrated turbo/Viterbi decoder is shown in Fig. 5.1. The shared components are represented with gray blocks. A specified input is used to switch the operating mode of the proposed design. While the turbo mode is activated, the components for Viterbi mode are all disabled by gated clock and vice versa. This will guarantee that redundant power consumption can be avoided in both operating modes.

According to the operating mode, the input data goes through the input cache or transition metric unit (TMU, also called branch metric unit or simply BMU) of Viterbi decoder (VD) for turbo mode and Viterbi mode, respectively. In turbo mode, the sliding windowed approach introduced in section 2.3 is adopted with a sub-block length of 20. The data output of the input cache will later go through three additional TMU for data preparation. The overall architecture consists of 24 ACS units, which are separated into 3 blocks to complete α, β_1, and β_2 recursions in parallel in Turbo mode. In Viterbi mode, only 16 of 24 ACS units are used for trellis decoding. The path metrics in both algorithms are obtained by accumulating branch metrics. Finally, the data output of ACS units may be imported into LLR computation unit to do iterative decoding for turbo mode or into path metric unit (PMU) of VD so that trace back can be done periodically in Viterbi mode.

In Fig. 5.1, the memory occupies a significant area of our design. It includes input cache, forward path metric storage of turbo decoder, and interleaver/de-interleaver memory shared with survivor memory in Viterbi mode. To save chip area, time-multiplexing method is utilized to provide double memory access frequency so that all memory blocks but input cache are implemented with single-port SRAM.

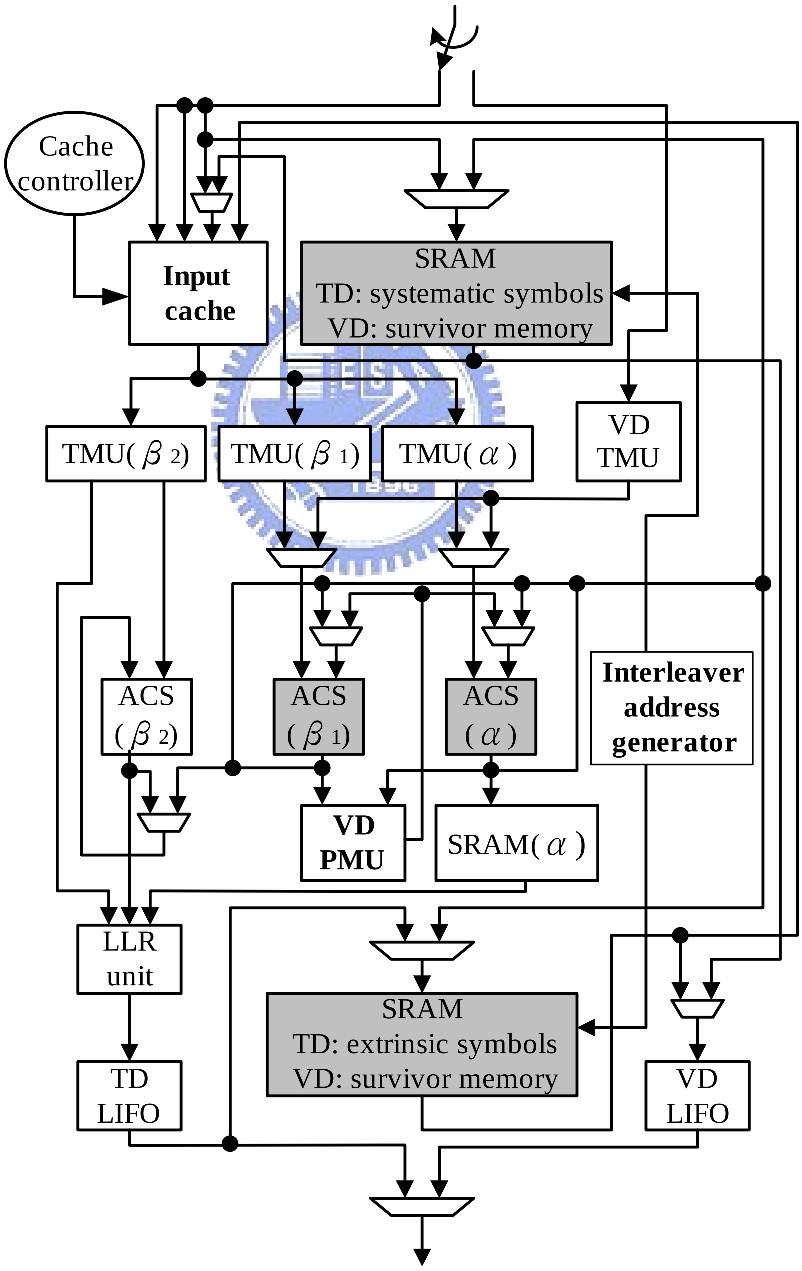


Fig. 5.1 The proposed architecture of integrated turbo/Viterbi decoder

5.2 Architecture of Turbo Decoder

The architecture of integrated turbo/Viterbi decoder operated in turbo mode is shown in Fig. 5.1, in which all disabled blocks and unconnected lines are represented by dotted lines. Although iterative decoding with ten iterations provides 0.2dB coding gain compared with that with six iterations, the former scheme is not adopted in our design due to its longer output latency and higher power dissipation. Detail operating flow is described as follows.

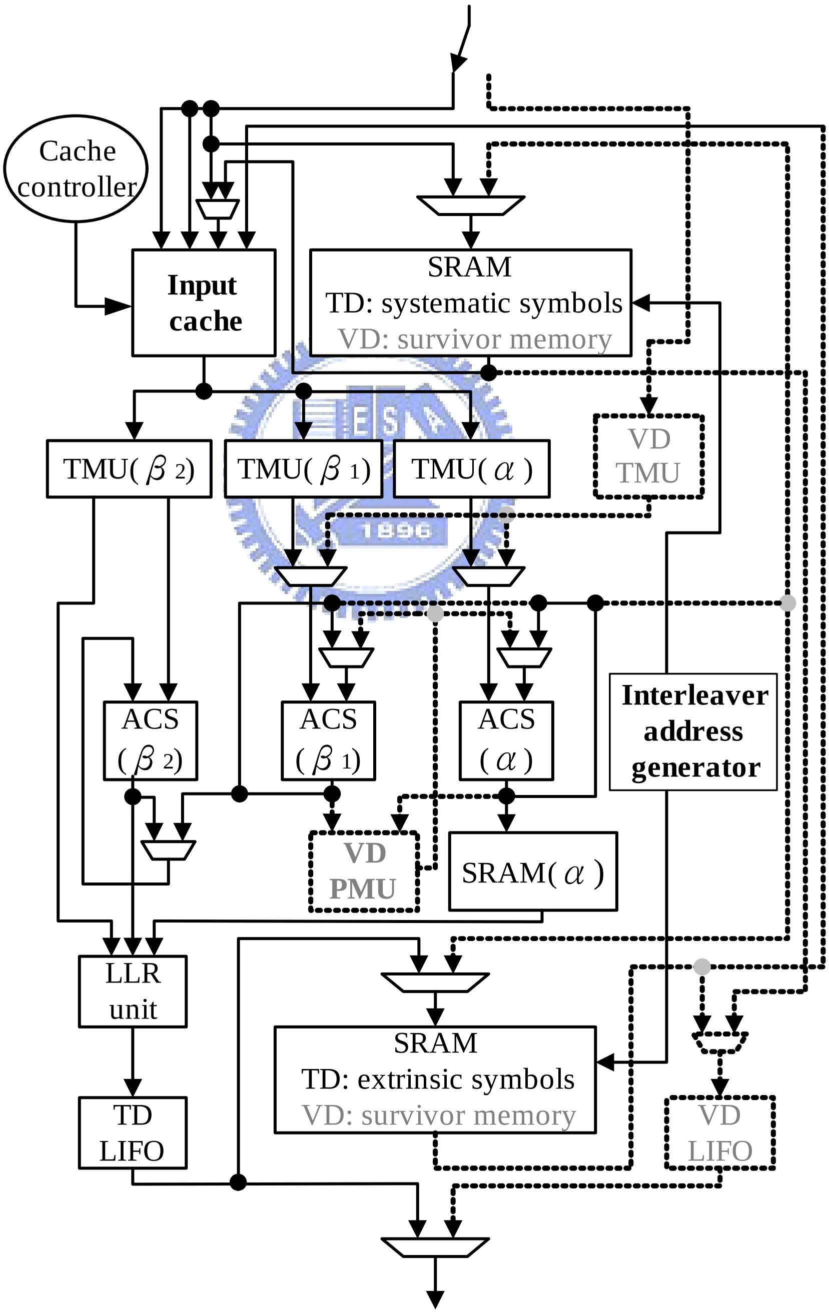


Fig. 5.2 The architecture of integrated turbo/Viterbi decoder in turbo mode

5.2.1 Single MAP Decoder design

In general, the block diagram of turbo decoder can be expressed as Fig. 2.4, which consists of two MAP decoders, two interleavers, and one de-interleaver. To implement the turbo decoder according to this diagram directly is too complicated and not efficient. Since two constituent decoders are identical, a single MAP decoder is proposed to not only reduce design cost but also simplify the control logic for two SISO decoders.

To achieve a single MAP decoder architecture, a full decoding iteration is split into two phases. In the first phase for the SISO decoder1, the MAP decoder reads systematic data, parity data and extrinsic values which come from the other decoder after de-interleaving. The output extrinsic data are stored in memory. As in the second phase for the SISO decoder2, the MAP decoder copes with permuted systematic data, parity data from the second encoder, and *a priori* values which are the interleaved extrinsic output from SISO decoder1. A simplified architecture of Fig. 2.4 is illustrated in Fig. 5.3. Note that there is an additional input cache and only one memory block for extrinsic data storage here. These will be introduced later in sub-sections 5.2.2 and 5.2.6 respectively.

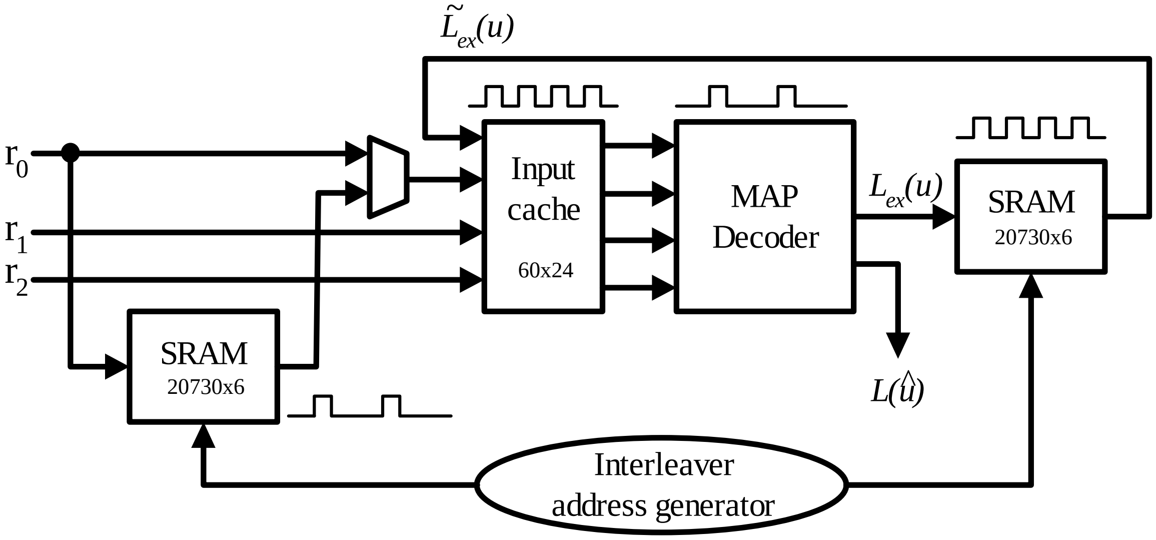


Fig. 5.3 A single MAP decoder architecture for turbo decoding

5.2.2 Cache design

As what we mentioned in the previous section, a sliding windowed approach is adopted in our design. Referring to Fig. 2.7, the data of each sub-block needs to be read three times by ACS-β_1, ACS-α, and ACS-β_2 units separately. Thus, an input cache is implemented to reduce repeated accesses of external memory, and power-down can also be achieved. The cache keeps three consecutive sub-blocks, and is equipped with one writing port for data updating and three reading ports for ACS units. As shown in Fig. 5.4, the cache is implemented by a dual-port SRAM with the size of 60x24 bits, and uses time multiplexed approach to provide four data ports. A set of additional registers is employed at output port-2 to guarantee that all outputs of the cache will be synchronized at the same clock rising edge. Detail timing chart is shown in Fig. 5.5.

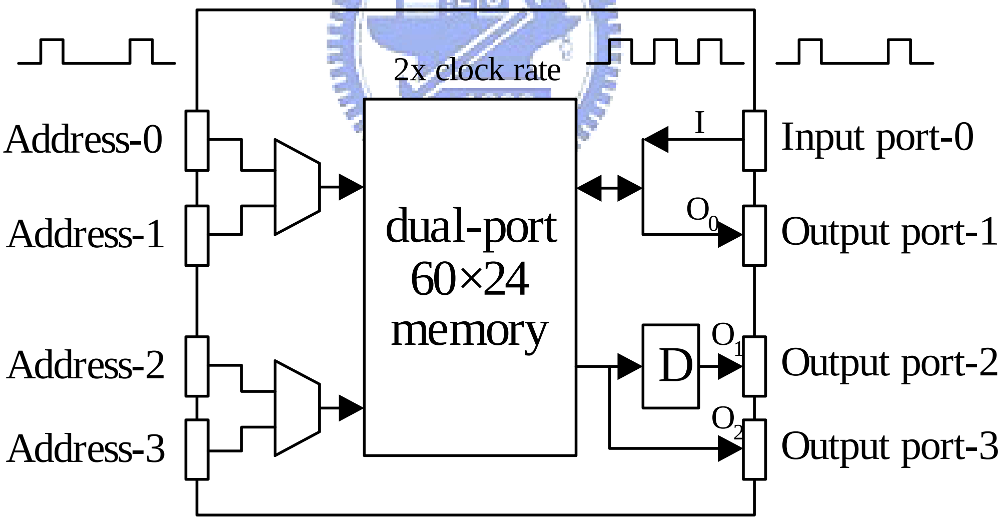


Fig. 5.4 The input cache architecture

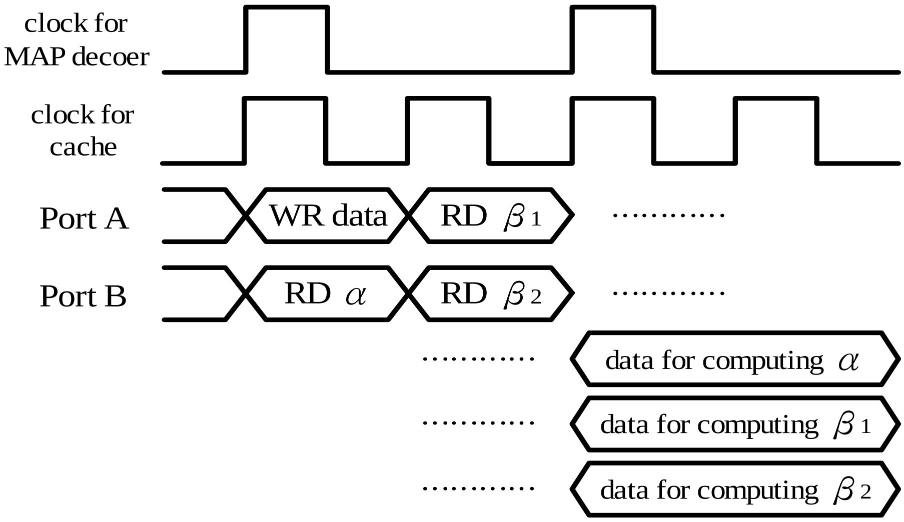


Fig. 5.5 The detail timing chart of the proposed input cache

5.2.3 Transition Metric Unit (TMU)

In 3GPP2 standard, eight branch metrics are required for LLR computation. According to the formula listed in (2. 14), each branch metric is obtained by adding or subtracting received symbols and *a priori* data together depending on the branch codewords. Implementing it directly will consume lots of adders and subtractors. A simple method to overcome this problem is to use an equivalent formula listed in (5. 1)

$$\gamma_k(s_{k-1},s_k)=\left(L_c r_{k,1}+L(u_k)\right)\cdot u_k+\sum_{v=2}^{n} L_c r_{k,v}\cdot x_{k,v} \tag{5. 1}$$

where $u_k=\{0,1\}$. By multiplying with u_k=0, some terms can be removed without changing the difference of branch metrics. The modified architecture of TMU is shown in Fig. 5.6.

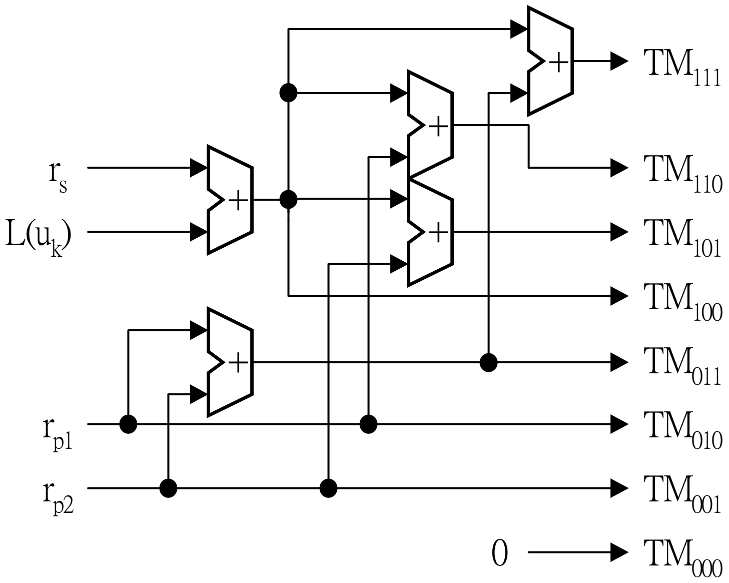


Fig. 5.6 The TMU architecture for turbo decoder

5.2.4 ACS Unit

The trellis diagram of both turbo code and convolutional code can be decomposed into many basic butterfly units. Each one can be implemented by the ACS units. Due to accumulating the branch metrics, the normalization is necessary to prevent error due to overflow. Several methods of the normalization had been developed [20]. These include reset, variable shift, fixed shift, and modulo normalization [21]. In our proposed design, the last one scheme is adopted. The key idea of the modulo normalization is not to avoid the overflow, but to accommodate the overflow. Therefore, it can rescale the path metrics locally, and can be implemented by the 2’s complement adders. The penalty of the modulo normalization is that the extra one bit is required for all components in each ACS unit. Compared with other normalization schemes, its overhead is quite small.

The design of ACS unit should be compatible for both Max-Log-MAP algorithm and Viterbi algorithm. Referring to (2. 23) and (2. 24), we can easily find that both α and β recursions perform the same add-compare-select operations as that in Viterbi decoder. The

ACS architecture for dual mode turbo/Viterbi decoder is shown in Fig. 5.7. The detail bit-width information is also shown here.

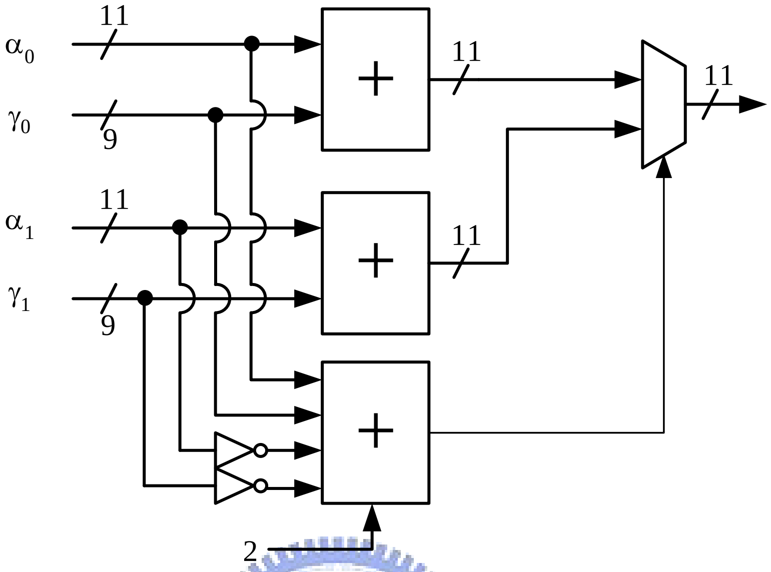


Fig. 5.7 The ACS architecture for dual mode turbo/Viterbi decoder

1896

5.2.5 LLR unit

LLR unit is a function-block that responses to compute *a posteriori* LLR and extrinsic information according to the path metrics and branch metrics. The proposed architecture is shown in Fig. 5.8. By gathering the forward path metrics from SRAM storing α, the backward path metrics from ACS-β_2 units and branch metrics from TMU-β_2, the LLR for each branch can be figured out in 16 LLR-unit cells. After taking the maximum LLRs for both u_k=+1 and -1, the *a posteriori* LLR can be obtained according to (2. 22). The extrinsic information is also acquired base on (2. 15). Responding to section 4.1.2, the extrinsic data should be bounded with 4.2 format for cost consideration. Any values exceeding the range will be pull back to +7.75 or -8.00 according to their sign bits. Finally, an additional Last-in-first-out (LIFO) is used for symbol re-ordering due to backward LLR computations.

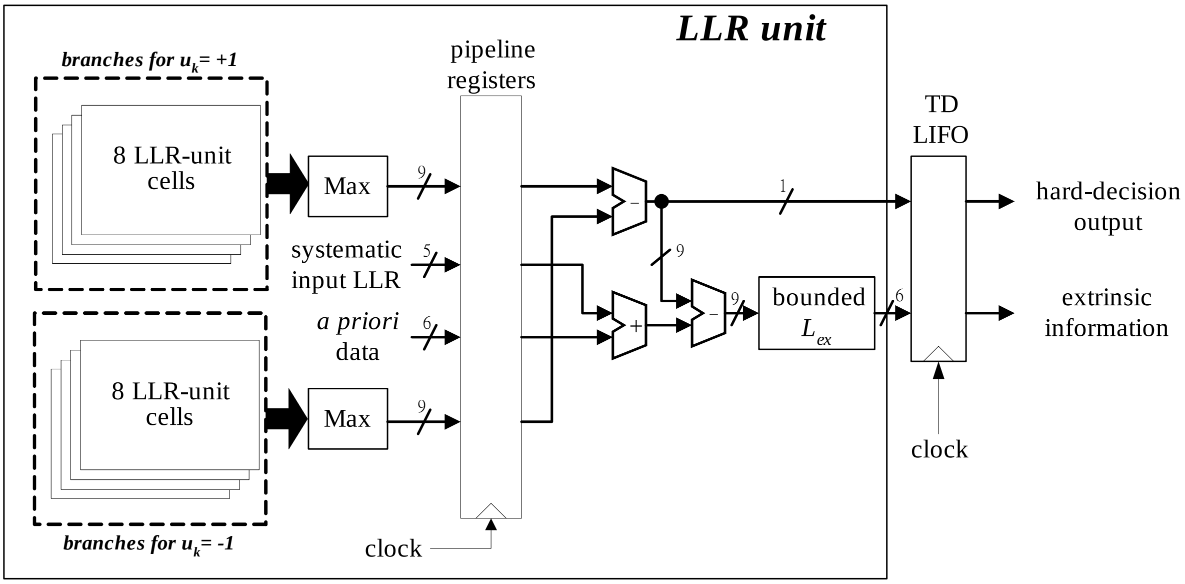


Fig. 5.8 The LLR unit architecture for turbo decoder

5.2.6 Interleaver design

The embedded interleaver/de-interleaver that supports a maximum block length of 20,730 is designed to reduce the amount of time required to permute symbols. Although interleaver and de-interleaver must co-exist in turbo decoder, memory sharing between them can be realized because of the single MAP decoder design. As what we mentioned in section 5.2.1, in the first phase for the SISO decoder1, the single MAP decoder reads *a priori* information from the memory in sequence. Once the extrinsic data is generated from the MAP decoder, it can be written into the memory in sequence, too. By the similar way, in the second phase for the SISO decoder2, the MAP decoder reads *a priori* information from the memory in permuted order. Once the extrinsic data is generated from the MAP decoder, it can be written into the memory in permuted order, too. The key point is that since the data is read first, there is no conflict while updating data. This idea can be illustrated with Fig. 5.9. In order to write and read data from memory at the same time, two memory blocks are required for the block interleaver in traditional architecture. However, this is a heavy burden for the chip implementation because of the large block length. To write and read data at the same time without increasing memory size, the time multiplexed approach is utilized to provide two data ports. Therefore, the clock rate for SRAM storing extrinsic symbols should be twice of that in the MAP decoder.

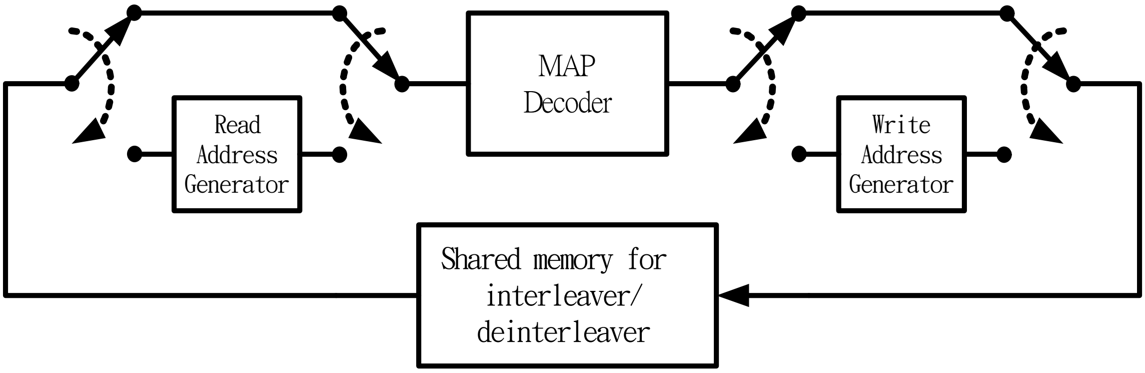


Fig. 5.9 The architecture of shared memory design in turbo decoder

The permutation realized by address management operates on the fly with the MAP decoder, which results in no additional delay within each iteration. Fig. 5.10 shows the address generator for interleaving operation. A large memory of address table with a size of 310.95kb can be eliminated by the on the fly address calculator. However, in 3GPP2 standard, the generator may produce invalid addresses and stall the MAP decoder, which will introduce redundant latency and thus decrease the throughput rate. This problem can be solved by a duplicated address generator since it is guaranteed that there must be at least one valid address among any two successive permuted addresses. While an invalid address is observed, the address from the other generator is adopted. Two SRAMs of 20,730 words are included in the proposed decoder to store systematic and extrinsic symbols respectively. And both of them are single-port structures to avoid the area overhead of multi-port memory.

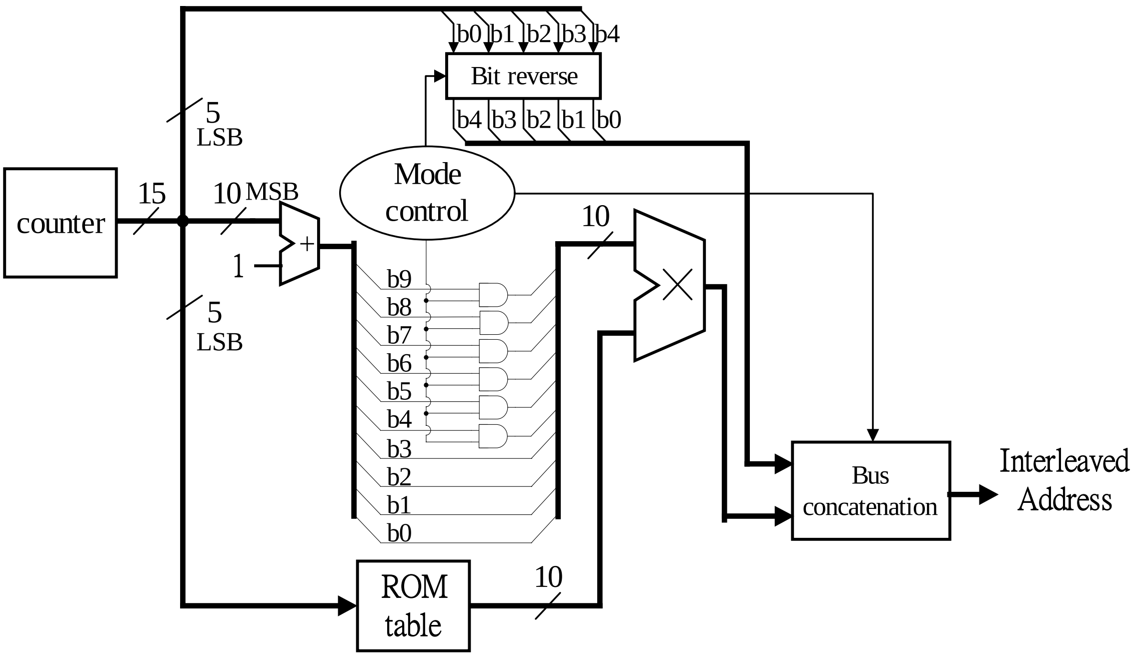


Fig. 5.10 The address generator for the interleaver of 3GPP2 turbo decoder

5.3 Architecture of Viterbi Decoder

In Viterbi mode, 256 states trellis decoding is implemented with 1/2, 1/3, 1/4, and 1/6 code rate. The architecture of the Viterbi decoder is based on the accomplished hardware components in the turbo decoder. Since the maximum throughput rate specified in 3GPP2 standard is not so critical, the fully parallel architecture is not necessary here. Fig. 5.11 shows the architecture of integrated turbo/Viterbi decoder operated in Viterbi mode. 16 of 24 ACS units included in ACS-α and ACS-β_1 are employed to finish 256 ACS operations within 16 cycles. The memory for interleaver in turbo decoder is treated as the survivor memory. Detailed operating flow is described as follows.

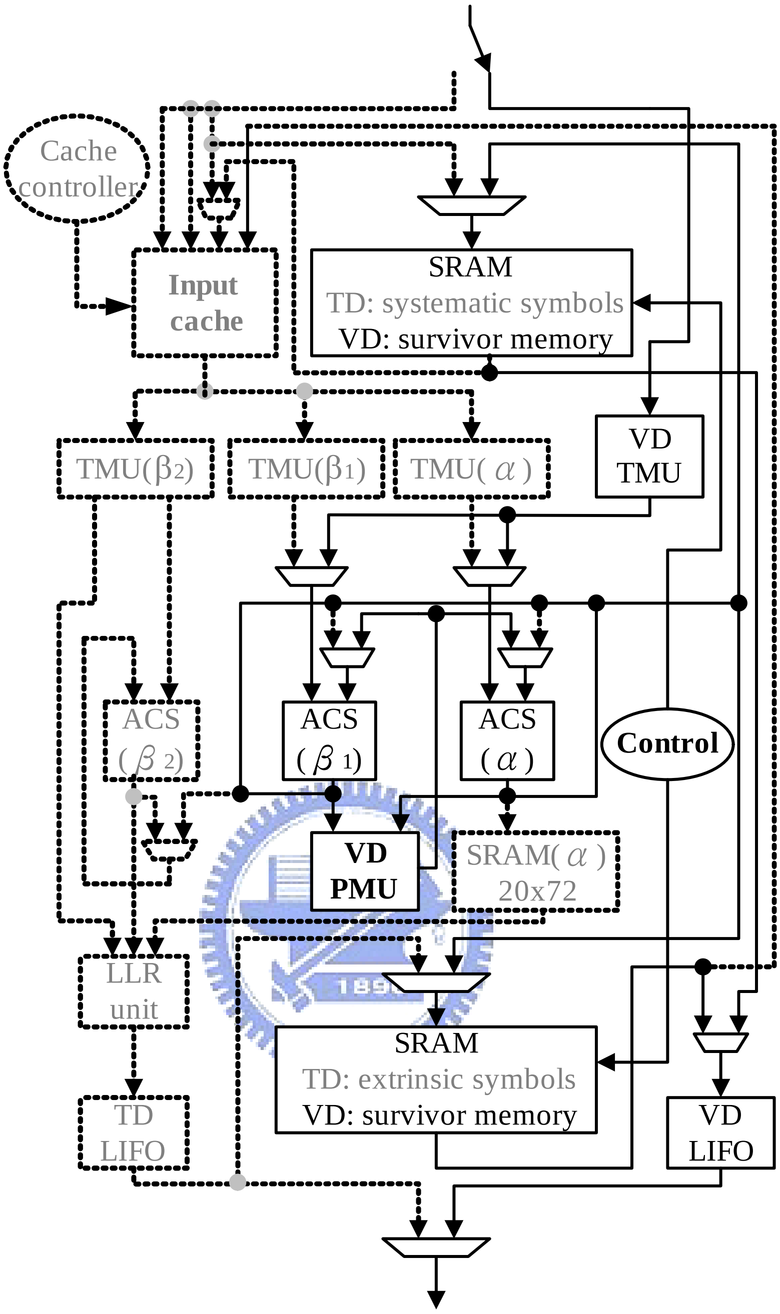


Fig. 5.11 The architecture of integrated turbo/Viterbi decoder in Viterbi mode

5.3.1 Transition Metric Unit (TMU)

Due to limited number of ACS units, the ACS operations for each time index in the trellis have to be separated into 16 cycles. Thus, different branch codewords may occur in every ACS unit according to the different cycle count. A TMU cell is designed for Viterbi decoding to deal with this problem. The architecture of TMU cell is shown in Fig. 5.12. The

TMU controller exports miscellaneous codewords base on current operating code rate and cycle count. After that, the transition metrics are obtained by accumulating the difference of soft-input symbols and codewords. In the TMU, there are 32 TMU cells assigned for 16 ACS units to complete all branch metric computations.

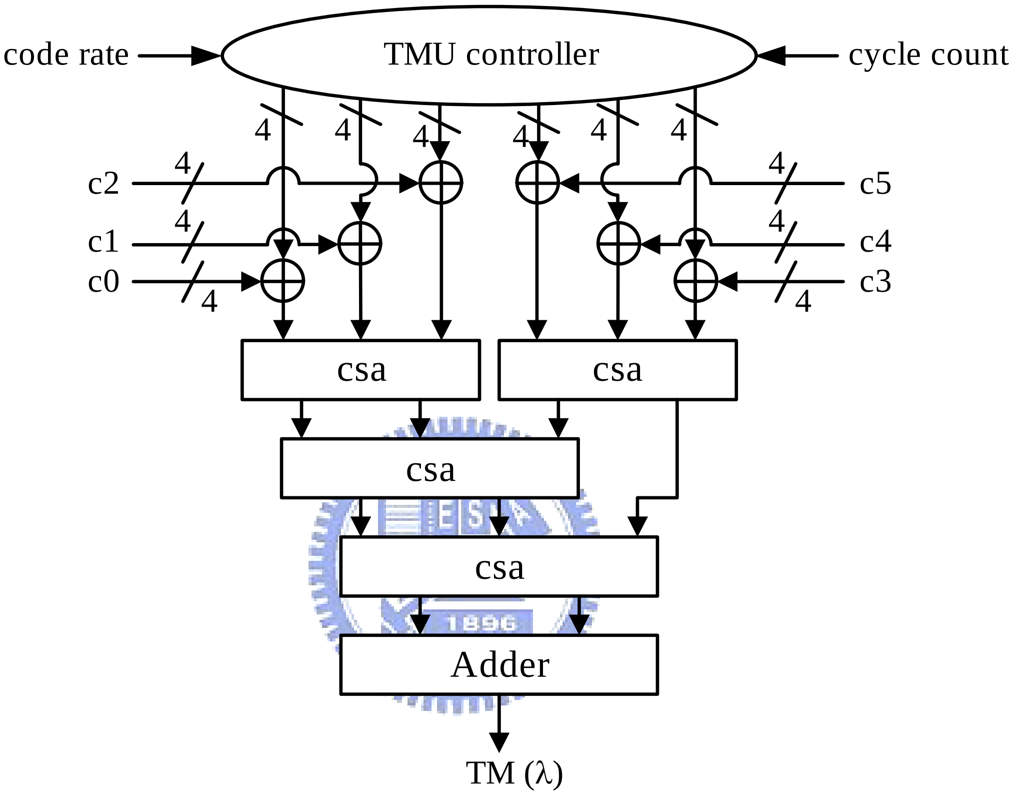


Fig. 5.12 The architecture of TMU cell

5.3.2 Survivor Memory Management

Several survivor memory management methods had been introduced in [22]. Among them, a modified 3-pointer even algorithm is employed to achieve high-speed trace-back operation. By this method, the amount of the survivor memory required must be triple of that of traditional scheme. These memory blocks are distinguished into six banks; each one has a length of L/2 where L is the truncation length. The basic idea is illustrated in Fig. 5.13. In this graph, one WRITE (WR), two TRACE-BACK (TB) and one DECODE (DC) operations are preceded in parallel. The detail descriptions are listed as follows.

- WRITE (WR)

  The 16 decision bits made by ACS units are written into survivor memory. For each cycle, the WRITE pointer is moved forwardly to avoid data conflicting. To complete 256-state decision bit selections, the WRITE operation is also divided into 16 cycles.

- TRACE-BACK (TB1 and TB2)

  The TRACK-BACK operation starts from the time index t=3L/2. Since each memory bank has a length of L/2, two TRACE-BACK operations must be performed in two memory banks to achieve trace-back method before decoding. In this step, the pointer is moved backwardly. The decision bit, D_t, is chosen from 256 survivor paths according to the method introduced in section 3.3. A track-backed state S_{t-1} is obtained by $S_{t-1} = (S_t <<1) | D_t$, where << denotes the left shift operation and | denotes the OR operations.

- DECODE (DC)

  An additional DECODE operation is used to finish Viterbi decoding after the completeness of TRACK-BACK in the previous bank. The action of DECODE operation is exactly the same as that of TRACK-BACK operation. Because of backwardly decoding, the decoding sequence is in reverse order and thus an additional LIFO buffer of length L/2 is required to perform bit re-ordering.

Based on the 3-pointer even algorithm, the modified architecture takes 16 cycles to WRITE, 2 cycles to TRACE-BACK, 1 cycle to DECODE, and thus totally 19 cycles to realize Viterbi decoding for each time section in trellis diagram. For clearness, the architecture of Survivor memory management is shown in Fig. 5.14.

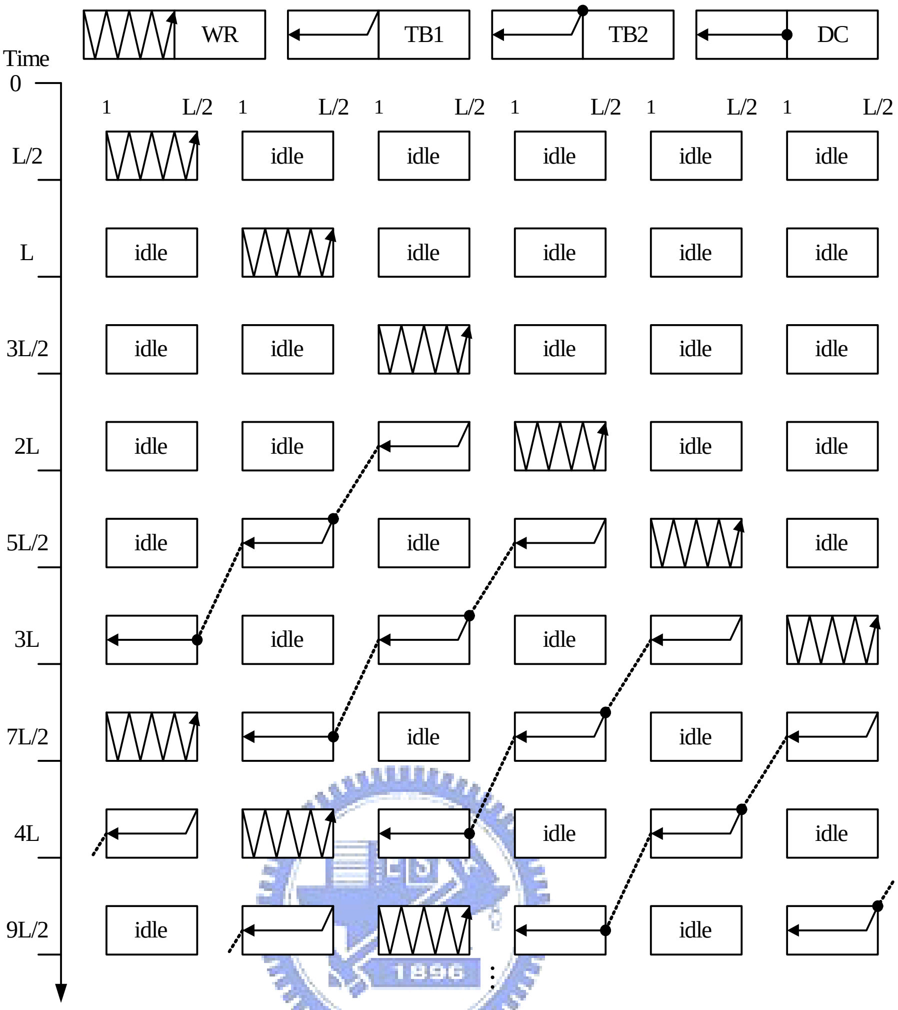


Fig. 5.13 The 3-pointer even algorithm for survivor memory management

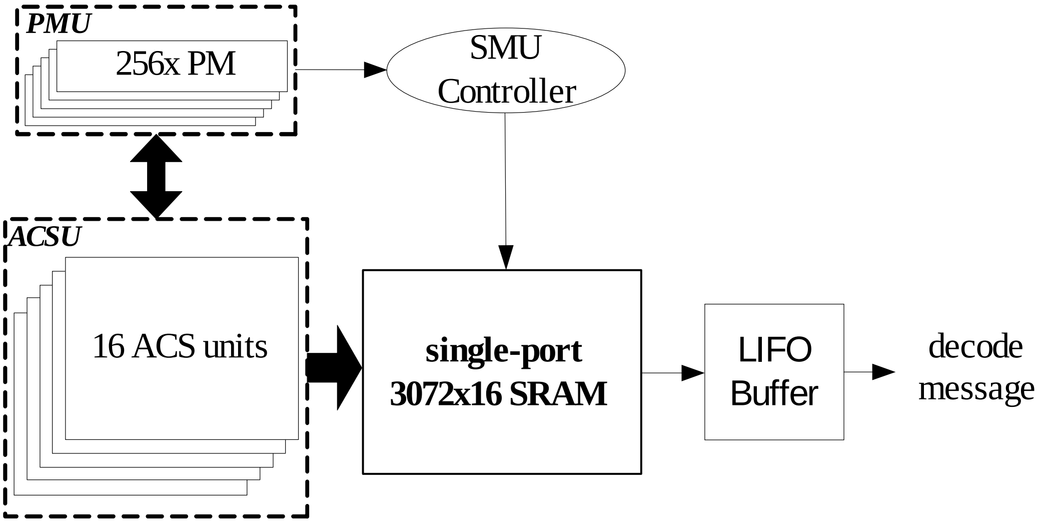


Fig. 5.14 The architecture of survivor memory management

5.4 *Summary*

In this chapter, we have presented the architecture of our proposed design and introduced all components for each mode separately. In turbo mode, the kernel of MAP decoder, including ACS units, can be operated at a lower clock rate due to two-phase clock design. A dual-port input cache is embedded to reduce times of external memory access. Both features make power consumption be even lowered down while operating in turbo mode. Moreover, the efficient interleaver design removes the redundant memory block employed in interleaver/de-interleaver. In Viterbi mode, a modified 3-pointer even algorithm is used to increase the throughput rate. Based on 16 shared ACS units, each decoded bit can be obtained every 19 cycles.

1896

Chapter 6
Chip Implementation

6.1 Chip specification

The decoder is implemented by the cell-based design flow, and fabricated in a 0.18 μm 1P6M standard CMOS process. In turbo decoding mode, two clock domains are used in memory and datapath respectively. The lower clock rate is achieved by clock gated from the input clock. The double clock rate provides the memory with higher bandwidth, and the single-port memory is sufficient in the proposed design except the cache memory. The chip size is 11.56mm^2 with the core size of 7.29mm^2. The total gate count is about 115k gates including the path metric memory for Viterbi decoder. Three single-port and one dual-port SRAM are embedded in the chip with a total size of 251.6kb. The maximum IR drop that occurs in MAP decoder is about 5.7mV while the supply voltage is 1.8V. This will assure the stable and correct operation. The detail chip specification is summarized in Table 6.1. The microphoto of the chip is shown in Fig. 6.1.

The chip has been tested and can work at 100MHz (50MHz in datapath) under 1.60 ~ 1.98V supply voltage, which can provide 4.52Mb/s for turbo decoding in six iterations and 5.26Mb/s for Viterbi decoding. Table 6.2 shows the power measurement result while decoding turbo codes and convolutional codes. Note that 1Mb/s is the average throughput rate in 3GPP2 standard.

Table 6.1 Summary of the decoder chip

Technology	0.18 μm CMOS
Package	84 CLCC
Supply voltage	1.8V core/ 3.3V IO
Chip size	11.56 mm^2
Embedded SRAM	251.64kb
Supported code rate	1/5 for turbo code 1/2, 1/3, 1/4, 1/6 for Viterbi mode
Maximum data rate	4.52Mb/s for turbo mode 5.26Mb/s for Viterbi mode

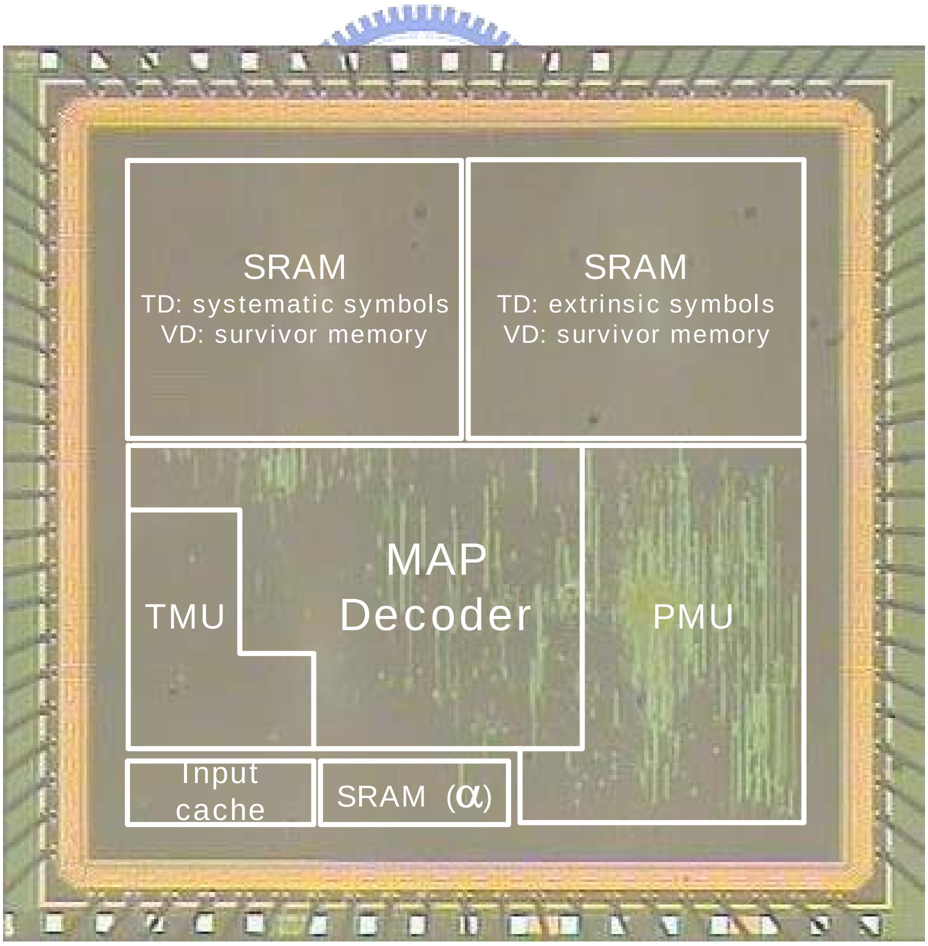


Fig. 6.1 The microphoto of the decoder chip

Table 6.2 Power measurement result of the decoder chip

Mode	Data rate	Clock rate	Block length	Power
Turbo mode	4.52Mb/s	100MHz	20,730	121 mW
	3.1Mb/s	66MHz	20,730	83 mW
		100MHz	378	81 mW
	1Mb/s	25MHz	20,730	29.5 mW
		33MHz	378	28.8 mW
Viterbi mode	5.26Mb/s	100MHz	Code Rate = 1/2	116.46 mW
	1Mb/s	20MHz	Code Rate = 1/2	25.1 mW

6.2 *Comparison with other similar work*

Up to now, there is no issue in *IEEE* publication about dual mode channel decoder implementation for 3GPP2 standard. Thus, we make a comparison with the unified channel decoder for 3GPP standard in [23] shown in Table 6.3. Due to lower code rate specified in both turbo mode and Viterbi mode, the gate count is larger than that of [23]. However, the proposed design is more economic in power dissipation as a whole.

Table 6.3 Comparison with other similar work

	The proposed design [26]	[23]
Technology	0.18 CMOS	0.18 CMOS
Compatible Standard	3GPP2	3GPP
Gate Count	115,000	85,000
Core Size	7.29 mm^2	9 mm^2
Supply Voltage	1.8V	1.8V
Maximum Clock Rate	100MHz	110.8MHz
Maximum Throughput Rate (Turbo Mode)	4.52 Mb/s with 6 iterations	2.5 Mb/s with 10 iterations 4.1 Mb/s with 6 iterations
Power (Turbo Mode)	83 mW@66MHz (3.1 Mb/s)	292 mW@88MHz (2.048 Mb/s)
Power (Viterbi Mode)	25.1 mW (1 Mb/s)	116.8 mW (1 Mb/s)
Embedded Memory Size	251.64 Kb	239 Kb

6.3 *Summary*

In this chapter, a chip implementation result is presented. The turbo decoding is able to achieve a maximum data rate of 4.52Mb/s with relative lower power consumption. The Viterbi decoding consists of 256 states and uses the same datapaths as those of the turbo decoder. Various coding rates, including 1/2, 1/3, 1/4, and 1/6, are supported. As compared to the unified channel decoder in where 292mW is required to decode a 2Mb/s data stream with ten iterations, the proposed design is more efficient in power dissipation.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

In this thesis, we talked about an implementation method for a dual mode Turbo/Viterbi decoder compatible for 3GPP2 standard. The chip is completed by verilog-HDL and UMC 0.18μm standard cell library. The chip die size is 3.40 x 3.40 mm^2 with the core size of 2.70 x 2.70 mm^2. It contains 115k gate counts of logic cell. The maximum iteration number for turbo decoding is fixed to six. With supply voltage of 1.8V, the power consumption in Turbo mode is about 83mW while working at 66MHz to achieve 3.1 Mb/s throughput rate; and that in Viterbi mode is about 25.1mW while working at 20MHz to achieve 1Mb/s throughput rate.

7.2 Future Work

Up to now, the early termination scheme is regarded as the most efficient way to reduce the power consumption in turbo decoders. It uses several characteristics in turbo decoding to judge if decoding sequence is nearly correct before maximum iteration number is achieved. Once iterative decoding can be stopped earlier, then the power can be saved. In [24], an iteration stopping criterion has been devised based on the cross entropy between the *a posteriori* probabilities of two SISO decoders for each iteration. In [25], two further simplified criteria were proposed for cost down. Although the performance of our design in the aspect of power consumption listed in Table 6.2 is attractive, there is no early termination technique used in it. This may be an aspect that we can try to better our design.

Bibliography

[1] Technical Specification Group Radio Access Network; Multiplexing and channel coding (FDD), 3GPP TS 25.212, V3.11.0, Sep 2002

[2] Physical Layer Standard for cdma2000 Spread Spectrum Systems, 3GPP2 Std. C.S0002-C, May 2002.

[3] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon Limit Error-Correcting Coding and Decoding: turbo-codes," in *IEEE Int. conf. Communications (ICC),* pp.1064-1070, May 1993.

[4] D.Divsalar, S. Dolinar, R. J. McEliece, and F. Pollara, "Performance Analysis of Turbo Codes," in *IEEE Military Communication conf.*, vol. 1, 5-8, pp. 91-96, Nov. 1995.

[5] J. Hagenauer and P. Hoeher, "A Viterbi Algorithm with Soft-decision Outputs and its Applications," in *IEEE GLOBE-COM*, Dallas, TX, pp. 47.1.1-47.1.7, Nov. 1989.

[6] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal Decoding of Linear Codes for Minimizing Symbol," in *IEEE Trans. Inform. Theory,* no.IT-20, pp 284-287, Mar, 1974.

[7] J. H. Andersen, "'Turbo' Coding for Deep Space Application," in *IEEE International Symposium on Inform. Theory*, 17-22, pp.36, Sep. 1995.

[8] J. H. Andersen, "Turbo codes extended with outer BCH code," in *Electronics Letters*, vol. 32, no. 22, 24, pp.2059-2060, Oct. 1996.

[9] P. Robertson, E.Villebrun and P. Hoeher, "A Comparison of Optimal and Sub-optimal MAP Decoding Algorithms operating in the Log Domain," *Proc. ICC'95*, Seattle, June 1995.

[10] J. Hagenauer, E. Offer, and L. Papke, "Iterative Decoding of Binary Block and Convolutional Codes," *IEEE Trans. Inform. Theory,* vol. 42, no. 2, pp. 429-445, Mar. 1996.

[11] J. A. Erfanian, S. Pasupathy, and G.Gulak, “Reduced Complexity Symbol Detectors with Parallel Structures for ISI Channels,” *IEEE Trans. Commun.*, vol. 42, no. 2/3/4, pp.1261-1271, Feb./Mar./Apr. 1994.

[12] T. A. Summers and S. G. Wilson, “SNR Mismatch and Online Estimation in Turbo Decoding,” *IEEE Trans. Commun.*, vol. 46, pp.421-423, Apr. 1998.

[13] A. Worm, P. Hoeher, N. Wehn, “Turbo-Decoding Without SNR Estimation,” *IEEE Commun. Letters*, vol. 4, no. 6, pp.193-195, June 2000.

[14] S. A. Barbulescu, “Iterative decoding of turbo codes and other concatenated codes,” University of South Australia, *PhD Dissertation*, Aug. 1995.

[15] S. A. Barbulescu, “On Sliding Window and Interleaver Design,” *Electronics Letters*, vol. 37, no. 21, pp.1299-1300, Oct. 2001.

[16] A. J. Viterbi, “Error bounds for convolutional codes and asymptotically optimum decoding algorithm,” *IEEE Trans. Inform. Theory*, vol. IT-13, no. 2, pp.260-269, Mar. 1973.

[17] Y. Wu and B. D. Woerner, “Internal data width in SISO decoding module with modular renormalization,” in *IEEE Vehic. Tech. Conf.*, vol. 1, pp. 675-679, May 2000.

[18] Y. Wu, B. D. Woerner, and T. K. Blankenship, “Data Width Requirements in SISO Decoding With Module Normalization,” in *IEEE Trans. On Commun.*, vol. 49, no. 11, pp. 1861-1868, Nov. 2001.

[19] T. K. Blankenship and B. Classon, “Fixed-Point Performance of Low-Complexity Turbo Decoding Algorithms,” in *IEEE Vehic. Tech. Conf.*, vol. 2 pp. 1483-1487, May 2001.

[20] C. B. Shung, P. H. Siegel, G. Ungerboeck and H. K. Thapar, “VLSI architectures for metric normalization in the Viterbi algorithm,” *IEEE International Conference on Communications*, vol. 4, pp.1723-1728, Apr. 1990.

[21] A. P. Hekstra, “An Alternative to Metric Rescaling in Viterbi Decoders,” *IEEE Trans. Commun.*, vol. 37, no. 11, pp. 1220-1222, Nov. 1989.

[22] G. Feygin and P. G. Gulak, “Architectural Tradeoffs for Survivor Sequence Memory Management in Viterbi Decoder,” *IEEE Trans. On Commun.*, vol. 41, no. 3, pp. 425-429, Mar. 1993.

[23] M. A. Bickerstaff, D. Garrate, T. Prokop, C. Thomas, B. Widdup, G. Zhou, L. M. Davis, G. Woodward, C. Nicol, R. H. Yan, “ A Unified Turbo/Viterbi Channel Decoder for 3GPP Mobile Wireless in 0.18-μm CMOS”, in *IEEE Journal of Solid-State Circuits*, vol.37, no.11, Nov. 2002

[24] M. Moher, “Decoding via Cross Entropy Minimization,” in *Proc. IEEE Globecom Conf.*, Houston, TX, Dec. 1993, pp.809-813.

[25] R. Y. Shao, S. Lin, and M. P. C. Fossorier, “Two Simple Stopping Criteria for Turbo Decoding,” *IEEE Trans. On Commun.*, vol. 47, no. 8, pp.1117-1120, Aug. 1999.

[26] C. C. Lin, Y. H. Shih, H. C. Chang, and C. Y. Lee, “A Dual Mode Channel Decoder for 3GPP2 Mobile Wireless Communications,” *Proc. 30th Eur. Solid State Circuits Conf. (ESSCIRC)*, 2004.

[27] 李鎮宜, 林建青, 施彥旭, “低複雜度之渦輪解碼器架構,” 中華民國專利申請中, 交通大學申請案編號: 04(專)A015

作者簡歷

姓名　：施彥旭

出生地：台灣省彰化縣

出生日期：1980.11.4

學歷：	1986.9 ~ 1992.6	彰化縣立鹿港國民小學
	1992.9 ~ 1995.6	彰化市私立精誠高級中學附設國中部
	1995.9 ~ 1998.6	彰化市私立精誠高級中學
	1998.9 ~ 2002.6	國立交通大學 電子工程系 學士
	2002.9 ~ 2004.6	國立交通大學 電子研究所 系統組 碩士

1896

得獎事績

九十一學年度 第二學期電子研究所書卷獎

九十二學年度 全國大專院校 FPGA 系統設計競賽 Xilinx 研究所組特優

九十三學年度 斐陶斐榮譽學會新榮譽會員

發表論文

- Yew-San Lee, Cheng-Mou Yu, Hung-Kuo Wei, Yen-Hsu Shih, Chen-Yi Lee, "**A novel DCT-based bit plane error resilient entropy coding scheme and codec for wireless image communication**," in *IEEE ISCAS*, May 2002.

- Hung-Kuo Wei, Yew-San Lee, Yen-Hsu Shih, Chen-Yi Lee, "**A novel fixed bit plane error resilient image coding for wireless multimedia transmission**," in *IEEE ICIP*, Sep. 2002.

- Chien-Ching Lin, Yen-Hsu Shih, Hsie-Chia Chang, and Chen-Yi Lee, "**A Dual Mode Channel Decoder for 3GPP2 Mobile Wireless Communications**," in *IEEE ESSCIRC*, 2004